

WL-TR-94-1102

MACHINE LEARNING: A COMPARATIVE
STUDY OF PATTERN THEORY AND C4.5



MR JEFFREY ALAN GOLDMAN

SYSTEMS CONCEPTS SECTION
MISSION AVIONICS DIVISION

JUNE 1994

FINAL REPORT FOR 12/01/93-06/01/94

DTIC
CITE
OCT 1 1994
D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

AD-A285 582
2
[Barcode]

DTIC QUALITY INSPECTED 2

AVIONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT PATTERSON AFB OH 45433-7409

2917
94-32533
[Barcode]

9433

30

NOTICE

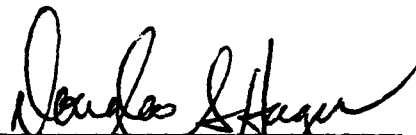
When Government drawings, specifications or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility nor any obligation whatsoever. The fact that the government may have formulated, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise in any manner construed, as licensing the holder or any other person or corporation, or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

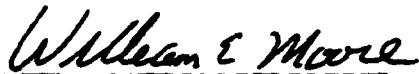
This technical report has been reviewed and is approved for publication.



JEFFREY ALAN GOLDMAN
Computer Scientist
Technology Section



DOUGLAS S. HAGER, Chief
Technology Section
Target Recognition Technology
Branch



WILLIAM E. MOORE, Acting Chief
Mission Avionics Division
Avionics Directorate

If your address has changed, if you wish to be removed from our mailing list or if the addressee is no longer employed by your organization, please notify WL/AART, Bldg 22, 2690 C Street STE 1, Wright-Patterson AFB, OH 45433-7408 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE JUNE 1994	3. REPORT TYPE AND DATES COVERED FINAL 12/01/93--06/01/94	
4. TITLE AND SUBTITLE MACHINE LEARNING: A COMPARATIVE STUDY OF PATTERN THEORY AND C4.5			5. FUNDING NUMBERS PE 61101F PR 0100 TA AA WU 13	
6. AUTHOR(S) MR. JEFFREY ALAN GOLDMAN				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AVIONICS DIRECTORATE WRIGHT LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT-PATTERSON AFB OH 45433-7409			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AVIONICS DIRECTORATE WRIGHT LABORATORY AIR FORCE MATERIEL COMMAND WRIGHT-PATTERSON AFB OH 45433-7409			10. SPONSORING/MONITORING AGENCY REPORT NUMBER WL-TR-94-1102	
11. SUPPLEMENTARY NOTES				
12a. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) The Machine Learning field has identified several different inductive bias classes with Occam's Razor being held as an accepted paradigm. C4.5, an extension of ID3, is one of the leaders in this class of learning systems with which other systems measure their ability. A completely different approach, yet still a method in the class of Occam biased learning mechanisms, is Pattern Theory. This approach seeks to recognize patterns in a robust manner using function decomposition. FLASH, the embodiment of Pattern Theory is itself, an inductive learning system. In this study, we hope to show that the Pattern Theoretic approach is not only as good as the classic decision tree methods, but also it exhibits strong promise to be a robust technique to identifying patterns. We will compare C4.5 and Pattern Theory against a special benchmark set of patterns intended to illustrate many types of potential concepts to be learned. The comparisons will be made by constructing learning curves for each system.				
14. SUBJECT TERMS MACHINE LEARNING, PATTERSON THEORY, SUPERVISED LEARNING, C4.5, OCCAM-BASED LEARNING			15. NUMBER OF PAGES 129	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

Accession For	
NTIS - GPO	<input checked="" type="checkbox"/>
DDIC - GPO	<input checked="" type="checkbox"/>
Unpublished	<input type="checkbox"/>
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Contents

1 Background Into The Pattern Theory Approach	1
2 The C4.5 System	2
3 Description of Benchmark Set	3
3.1 RANDOM	3
3.2 RANDOM MINORITY ELEMENTS	3
3.3 BOOLEAN EXPRESSION	3
3.4 VARIATION ON THE MONK PROBLEMS	4
3.5 STRING FUNCTIONS	4
3.6 IMAGES	4
3.7 SYMMETRIC FUNCTIONS	4
3.8 NUMERICAL FUNCTIONS	5
4 Experimental Design	5
5 Experimental Results	12
6 Analysis	12
7 Conclusion and Summary	13
8 Future Work	13
A Individual Learning Curves of Each Function for C4.5 and FLASH	17
B Listing of the Decomposition Plan	124

List of Figures

1 Function on four variables	1
2 Decomposed function on four variables	2
3 Generalisation Comparison	15

List of Tables

1 C4.5 trials with default options, varying the number of trees	6
2 C4.5 trials with window size 0, varying the number of trees	7
3 C4.5 trials with threshold size 0, varying the number of trees	8
4 C4.5 trials with window and threshold size 0, varying the number of trees	9
5 C4.5 trials with varying options	10
6 C4.5 trials with grouping options	11
7 C4.5's best options with FLASH's best	14

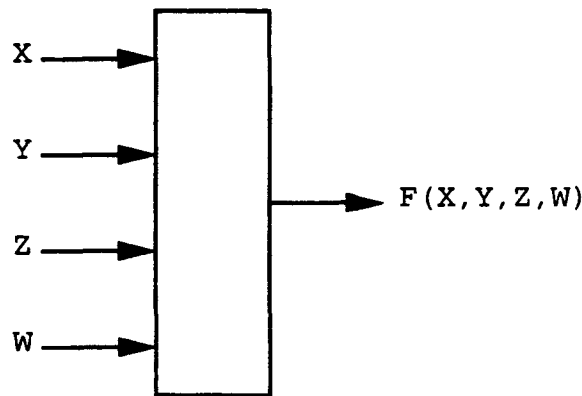


Figure 1: Function on four variables

1 Background Into The Pattern Theory Approach

The Pattern Theory¹ paradigm focuses on two central ideas shown in this section. The first is functions that the investigator wishes to learn, have low decomposed function cardinality. The second is functions with low decomposed function cardinality are learnable with a relatively small number of samples. In this section, we will present some background on function decomposition and how Pattern Theory uses this as a robust way to find patterns.

Decomposing a function involves breaking it up into smaller subfunctions. These smaller functions are further broken down until all subfunctions will no longer decompose. For a given function, the number of ways to choose two sets of variables (the partition space) is exponential. The decomposition space is even larger, since there are several ways the subfunctions can be combined and there are several levels of subfunctions possible. The complexity measure that we use to determine the relative predictive power of different function decompositions is called Decomposed Function Cardinality (DFC).

DFC is calculated by adding the cardinalities of each of the subfunctions in the decomposition. The cardinality of an n -variable binary function is 2^n . We illustrate the measure in the above figures. In Figure 1, we have a function on four variables with cardinality $2^4 = 16$. In Figure 2, we show the same function after it has been decomposed. The DFC of this representation for the original function is $2^2 + 2^2 + 2^2 = 12$. The DFC measures the relative complexity of a function. When we search through the possible decompositions for a function, we choose one with the smallest DFC. This decomposition is our learned concept.

The decomposed representation of the function is one that exhibits more information than the alternative. For example, Figure 1 is essentially a lookup table of inputs and outputs. Figure 2, on the other hand, is a function that is not simply a table. The decomposition, for example, could be two simple functions combined together.

Throughout the paper when we refer to a minimal function decomposition, we use "minimal" to mean a decomposition such that the DFC is the smallest possible for the entire set of decompositions. It is noted that a given minimal decomposition is not unique. For a more rigorous explanation of the inner workings of function decomposition or function extrapolation, the reader is referred to [1], [2] and [8].

An important point is that a function with a low DFC has been experimentally and theoretically determined to be learnable with a small number of samples [8]. Also, functions we are interested in learning, (i.e., functions that are highly "patterned,") have a low DFC. The Function Learning And Synthesis Hot-Bed (FLASH) was developed to explore function decomposition, and pattern finding. This paper will show that the FLASH program exhibits promising results for finding patterns robustly.

¹System Concepts, Wright Laboratory, WL/AART-2 2890 C Street STE 1, Wright-Patterson AFB, Ohio 45433-7408 Email: goldmanj@sa.wpafb.af.mil

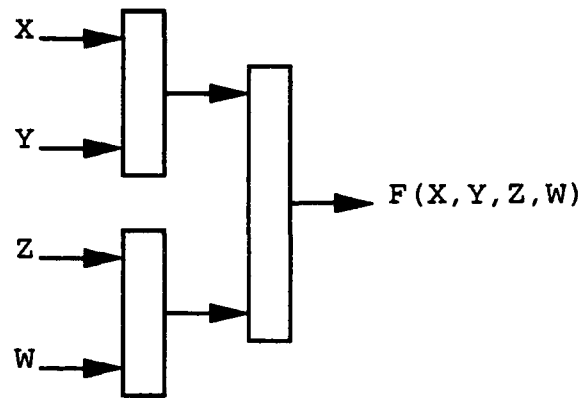


Figure 2: Decomposed function on four variables

2 The C4.5 System

C4.5 is a machine learning software package. A detailed study of it is given in [7]. The intention of this section is to familiarise the reader with general information about how C4.5 learns a concept. It is important to note that C4.5 is equipped to handle noisy data, conflicting data, continuous variables, and other features which are not our primary concern in this discussion. Although pattern theory is interested in these issues, we are testing performances given binary variables and 100% truthful data.

C4.5 is a decision-tree and rule-based system. This makes it a shallow reasoner. In other words, a deep understanding of the world is not required. The advantages of a shallow reasoner are in the separation of knowledge and control, there is a natural mapping to rules, the rules are modular, and it is easy to provide an explanation. The disadvantages are the brittleness associated with an implicit domain model, it lacks common sense, it lacks robustness, there are problems with formal verification, and often they have limited learning capabilities. A rule-based system works best with diagnosis, configuration and control, and process control. Moreover, rule-based systems are excellent for any system that has independent states, simple control flow (limited branching), and the ability to state knowledge needed without stating how it was obtained.

The C4.5 system has many different options that can be altered by the user for a given learning environment. The default options are as follows. First, given a training set, C4.5 builds a decision tree using the gain ratio criterion. In short, C4.5 chooses a test for the tree if it splits the training data into two unbalanced groups (i.e., Only Positive, Only Negative, Largest Positive). The measure is also normalized. In essence, the gain metric is a measure of Entropy. Second, C4.5 has a threshold default of 2 for a given test in a tree. The test must have at least 2 outcomes with a minimum number of cases. To be more precise, the sum of the weights of the cases for at least two subsets must attain a minimum of 2. We would increase this value if we had noisy data.

Other flexibility built into C4.5 includes changing the amount of pruning of the decision tree (for more generalisation and better predictability with noisy data), allowing C4.5 to choose among n best trees, windowing, debugging, use of continuous variables, using the older unnormalised gain, and various options for the rule induction program. For our purposes, we will not be concerned with pruning since we are interested in C4.5's best classification of the training data. We will, however, test different weight minimums for the gain metric, vary the number of trees, test grouping, and change minimums and maximums for windowing sizes.

Windowing in C4.5 is a feature that is used when creating the initial tree in the test cases. The procedure is to select a random number of training cases and build a tree. This tree is then used to classify the remaining training cases. Any misclassifications are used in a new refinement of the original tree. The cycle is repeated until a tree is built that correctly classifies all of the training data. C4.5 allows you to alter the number of cases to be included in the initial window. It also lets you specify a maximum number of cases that can be added to the window at each iteration. The grouping option for C4.5 allows the method to group discrete attributes by value. Quinlan describes this procedure in detail when we have discrete variables with many values. The purpose of grouping is to prevent forced binary splits. It uses an iterative merging technique on the training elements. We were uncertain at the time of testing, if this grouping would have any relevance to our binary variable domain.

Therefore, to be thorough, it was better to try the method than to ignore it.

It is useful at this point to discuss the option that allows C4.5 to build several trees retaining the best. The reason C4.5 doesn't produce an optimal tree (optimal in the sense that it is the smallest decision tree possible, consistent with the training set) every time is because this problem is NP-complete [5]. Thus, the gain metric is only a heuristic to build a near optimal tree in polynomial time.

Some shortcomings of C4.5 are mentioned in [7]. One is that C4.5 cannot correctly classify cases in which there are non-rectangular regions. For example, in dealing with continuous variables on a two-dimensional plane, the line $y = x(x, y > 0)$ does not lend itself to building rectangular regions. Instead, the triangular regions are approximated. Problems arising related to this are poorly delineated regions and fragmented regions. The author attributes fragmented regions to not having enough data to correctly classify.

3 Description of Benchmark Set

Our benchmark set of functions will be used to compare the learning ability of C4.5 and Pattern Theory. This set of functions, although not exhaustive, is designed to include many types of relationships that we might be interested in. The overall goal of testing on several different functions is to compare robust learning ability in this restrictive domain of binary variables. However, it is important to point out that although Pattern Theory is not yet equipped to handle continuous variables, the underlying theory generalizes to discrete and continuous variables. The reader is invited to a formal proof and further reading in [8]. The point is key since the binary domain is so restrictive.

The benchmark set includes some 4 dozen functions. The categories break down into: Boolean Expressions, String Functions, Images, Symmetric Functions, Numerical Functions, and Random Functions. All of the functions are of the form $F : [0, 1]^8 \rightarrow [0, 1]$. In other words, there are eight binary variable inputs and one binary variable output. A detailed description of each function is given here.

3.1 RANDOM

There are 3 functions that were randomly generated from FLASH with seeds 1,2, and 3. They are labeled: rnd1, rnd2, and rnd3.

3.2 RANDOM MINORITY ELEMENTS

There are 5 functions generated which have a fixed number of minority elements placed at random. The seed for each was 1. They are labeled: rnd_m1, rnd_m5, rnd_m10, rnd_m25, rnd_m50.

3.3 BOOLEAN EXPRESSION

These 10 KDD functions were designed to represent concepts in a database. KDD stands for Knowledge Discovery in Databases. They were first used in [3] and later in [4].

$$KDD1 = (x_1 x_3) + \bar{x}_2$$

$$KDD2 = (x_1 \bar{x}_2 x_3)(x_4 + \bar{x}_6)$$

$$KDD3 = (\bar{x}_1 + \bar{x}_2) + (\bar{x}_1 x_4 x_6)$$

$$KDD4 = \bar{x}_4$$

$$KDD5 = (x_1 x_2 \bar{x}_4) + (x_3 \bar{x}_5 x_7 x_8) + (x_1 x_2 x_5 x_6 x_8) + (\bar{x}_3 \bar{x}_5)$$

$$KDD6 = x_2 + x_4 + x_6 + x_8$$

$$KDD7 = (x_1 x_2) + (x_3 x_4) + (x_5 x_6) + (x_7 x_8)$$

$$KDD8 = (x_1 \bar{x}_2) \text{ XOR } (x_1 x_5)$$

$$KDD9 = (x_2 \text{ XOR } x_4)(\bar{x}_1 \text{ XOR } (x_5 x_7 x_8))$$

$$KDD10 = (x_1 \Rightarrow x_4) \text{ XOR } (\bar{x}_7 \bar{x}_8 (x_2 + x_3))$$

multiplexer, mux6, used in Kosa, this is a 2-address bit, 4-data bit multiplexer with two vacuous variables (x_0 and x_1) to make 8 inputs. Generated mux6 with FLASH and then edited to make mux8. 3/29/94.

"Deep functions" generated by Mike Noviskey: 04.26.94 and_or_chain8, (removed or_and_chain8 because or_and_chain8(x) = not(and_or_chain8(not(x))), as in DeMorgan's Theorem)

3.4 VARIATION ON THE MONK PROBLEMS

These are 8 binary variable approximation to the Monk's problems of [9].

x_1 : head shape (rnd, octagonal)

x_2 : body shape (rnd, octagonal)

x_3 : smiling (yes, no)

x_4, x_5 : holding (sword, balloon, flag, M16)

x_6, x_7 : jacket color (red, yellow, green, blue)

x_8 : has tie (yes, no)

monkish1: head shape equals body shape or jacket is red.

monkish2: exactly 2 of 6 attributes have 1st value.

monkish3: (jacket green & has sword) or (jacket not blue and body not oct.) generated with FLASH, 4/6/94.

3.5 STRING FUNCTIONS

These functions are operators on 8-bit binary strings. palindrome acceptor; pal, from FLASH 2/18/94.

palindrome output; pal_output, from Mike Noviskey, and PVWave, randomly generated 128 bits then mirror imaged them to create the outputs of an 8 variable function. 3/25/94

doubly palindromed output; pal_dbl_output, from Mike as above except he generated 64 bits and flipped them twice. 3/25/94

2 interval acceptors from FLASH 2/18/94;

interval1 accepts strings with 3 or fewer intervals

interval2 accepts strings with 4 or fewer intervals

2 sub-string detectors from FLASH 2/18/94;

substr1 accepts strings with the sub-string "101"

substr2 accepts strings with the sub-string "1100"

3.6 IMAGES

These functions are various bit maps. chXfY means character X from font Y of the Borland font set. All were generated with the Pascal program charfn.exe of 2/28/94.

ch8f0 - kind of a flat plus sign

ch15f0 - an Aztec looking design

ch22f0 - horizontal bar

ch30f0 - solid isosoles triangle

ch47f0 - slash

ch176f0 - every other column of a checker board

ch177f0 - checker board

ch74f1 - triplex J

ch83f2 - small S (thin strokes)

ch70f3 - sans serif F

ch52f4 - Gothic 4

3.7 SYMMETRIC FUNCTIONS

These functions are symmetric, meaning re-arranging the order of the inputs does not affect the output.

parity, from FLASH 2/22/94. contains_4_ones, ($f(x)=1$ if and only if the str x has k ones), from FLASH 3/2/94.
majority_gate, ($f(x)=1$ if and only if x has more 1's than 0's, from FLASH 3/2/94.

3.8 NUMERICAL FUNCTIONS

These functions are various arithmetic operators.

addition; add0, add2, add4 - outputs bits of a 4 bit adder, 0 is the most significant bit, generated with FLASH 2/22/94.

greater_than: $f(x_1, x_2) = 1$ if and only if $x_1 > x_2$, generated with FLASH 3/2/94.

subtraction: subtraction1, subtraction3 - output bits 1 and 3 of the absolute value of a 4 bit difference. 0 is most significant bit. generated with FLASH 3/2/94.

modulus2, output bit 2 of 4-bit modulus 0 is the most significant bit, generated with FLASH 2/22/94.

remainder2, output bit 2 of 4-bit remainder 0 is the most significant bit, generated with FLASH 2/22/94.

4 Experimental Design

The overall design of our experiment is as follows. First, several options were tested on all the benchmark functions in order to determine what parameters yielded the best performance for C4.5. Next, the resulting learning curves were compared with Pattern Theory.

The tests on the individual functions were as follows. First, each method was given a random set of data to train on ranging from 25 to 250 out of a total of 256 possible cases. Once the method was trained, the entire 256 cases were tested and the number of differences were recorded as errors. This procedure was repeated 10 times for a given sample training size in intervals of 25 yielding a maximum, minimum, and average number of errors for each. Thus, the total number of runs for each function was 100 of varying sample size. None of the learning was incremental. All of the runs were independent.

Our first task was to find the best options to maximise C4.5's performance over the entire training set. The options that were varied on C4.5 include the weight (threshold value for branching), initial windowing size, maximum window size, grouping, and the number of trees grown. The results are displayed in Tables 1-6. The rows are for each function tested. The columns are a description of the options given. The value in the table is the average number of errors for a given run, for a given function over the entire sampling of 25 to 250 samples (the average of all 100 points). The value at the bottom is the average number of errors for a given set of options over the entire set of functions. The smaller the number here, the better the overall performance.

The data is divided into six separate tables. Table 1 shows the relative performances of C4.5 with all of the default options, varying the number of trees. The first column is one tree (the default), the second column is 10 trees, and the third column is 100 trees. Table 2 shows the relative performance of C4.5 with the default options except that the windowing size (-w) is set to 0. Again, the three columns vary the number of trees from 1 to 10 to 100. Table 3 has all of the default options except the threshold parameter (-m) is set to 0. Once again, the number of trees are 1, 10, and 100 respectively. Table 4, like the others, has all the default options except the threshold is set to 0 and the window size is set to 0.

Table 5 is slightly different. The first column tests all of the default options with the threshold set to -1. This was compared with the identical run where the threshold was set to 0 (column 1 of Table 3) to ensure that 0 was indeed the lowest possible setting for the threshold parameter. The second column has the default options except the threshold is set to 0 and the maximum number of allowable cases is 256 (-i 256). This means essentially that since this is only an eight variable function, we have no imposed maximum number of cases. Column three is the default options except the threshold is set to 1 and the number of trees built is 10.

Finally, Table 6 shows some final experiments using grouping (-s) in addition to our best options given so far. Column 1 is the default options except the threshold is 0 and the number of trees built is ten. Column 2 is the same except the threshold value is 1.

If we examine all the tables in detail, we can conclude that having the threshold value set to 0 (smallest possible) or 1, there is a significant decrease in the number of errors when compared with the default. There does not appear to be any significant difference between a threshold of 0 and 1. As far as changing window size parameters, there is no significant change. In fact, once we use 10 or more trees, the initial window size parameter does not make any difference. The grouping option does not appear to give any advantage either for this data set. As far as the best number of trees, clearly more is better. However, there doesn't appear to be

Function Name	C4.5 Default	C4.5 10 trees	C4.5 100 trees
add0	22.56	22.62	23.12
add2	64.12	33.5	31.76
add4	39.72	3.38	3.38
ch15f0	47.96	37.92	35.62
ch176f0	20.54	6.4	5.44
ch177f0	39.14	2.08	2.08
ch22f0	29.96	15.18	14.6
ch30f0	18.18	16.06	15.72
ch47f0	33.08	26.5	26.2
ch52f4	30.76	28.78	27.96
ch70f3	15.5	15.08	14.72
ch74f1	20.9	20.12	19.66
ch83f2	33.42	32.24	32.42
ch8f0	20.84	16.84	16
contains_4_ones	80.18	80.6	80.84
greater_than	21.36	20.96	21.04
interval1	44.94	44.14	44.58
interval2	62.82	63	62.14
kdd1	0.64	0.64	0.64
kdd10	25.54	23.8	23.52
kdd2	3.76	3.84	3.6
kdd3	2.56	1.76	1.76
kdd4	0	0	0
kdd5	15.02	14.24	12.96
kdd6	3.84	3.36	3.16
kdd7	26.94	28.3	28.46
kdd8	16.32	6.04	3.2
kdd9	30.56	16.94	16.2
majority_gate	48.68	49.16	49.56
modulus2	16.72	17.42	17.16
mux8	23.28	17.8	14.52
pal	18.8	19.16	19.16
pal_dbl_output	73.74	68.7	65.14
pal_output	84.74	83.28	82.76
parity	128	128	128
remaindr2	31.95	31.49	31.23
rnd_m1	1	1	1
rnd_m10	11.02	11.24	11.24
rnd_m25	29.34	30	30
rnd_m5	5.74	5.52	5.52
rnd_m50	54.4	55.14	55.08
rnd1	87.84	86.46	85.68
rnd2	89.04	86.32	85.44
rnd3	85.3	84.34	83.68
substr1	41.44	37.38	36.22
substr2	33.3	29	26.68
subtract1	64.14	53.02	52.94
subtract3	39.72	3.38	3.38
Average over all Functions	36.2364583	30.87770833	30.34104167

Table 1: C4.5 trials with default options, varying the number of trees

Function Name	C4.5 1 tree Window Size = 0	C4.5 10 trees Window Size = 0	C4.5 100 trees Window Size = 0
add0	23.2	22.62	23.12
add2	45.54	33.5	31.76
add4	7.06	3.38	3.38
ch15f0	41.68	37.92	35.62
ch176f0	8.36	6.4	5.44
ch177f0	7.2	2.08	2.08
ch22f0	19.46	15.18	14.6
ch30f0	17.54	16.06	15.72
ch47f0	31.52	26.5	26.2
ch52f4	30.04	28.78	27.96
ch70f3	15.62	15.08	14.72
ch74f1	20.7	20.12	19.66
ch83f2	32.72	32.24	32.42
ch8f0	18.3	16.84	16
contains_4_ones	80.84	80.6	80.84
greater_than	20.42	20.96	21.04
interval1	44.32	44.14	44.58
interval2	62.74	63	62.14
kdd1	0.64	0.64	0.64
kdd10	24.72	23.8	23.52
kdd2	4	3.84	3.6
kdd3	3.04	1.76	1.76
kdd4	0	0	0
kdd5	15.92	14.24	13.96
kdd6	3.36	3.36	3.36
kdd7	27.2	28.3	28.46
kdd8	9	6.04	3.2
kdd9	20.18	16.94	16.2
majority_gate	48.62	49.16	49.56
modulus2	16.98	17.42	17.16
mux8	20.4	17.8	14.52
pal	19.16	19.16	19.16
pal_dbl_output	71.1	68.7	65.14
pal_output	84.5	83.28	82.76
parity	128	128	128
remaindr2	32.62	31.49	31.23
md_m1	1	1	1
md_m10	11.24	11.24	11.24
md_m25	30	30	30
md_m5	5.52	5.52	5.52
md_m50	54.92	55.14	55.08
md1	87.14	86.46	85.68
md2	88.04	86.32	85.44
md3	85.28	84.34	83.68
substr1	39.5	37.38	36.22
substr2	32.42	29	26.68
subtract1	58.38	53.02	52.94
subtract3	7.06	3.38	3.38
Average over all Functions	32.44166667	30.87770833	30.34104167

Table 2: C4.5 trials with window size 0, varying the number of trees

Function Name	C4.5 1 tree Threshold = 0	C4.5 10 trees Threshold = 0	C4.5 100 trees Threshold = 0
add0	15.93	16.26	16.58
add2	44.78	27.4	26.32
add4	27.29	3.6	3.6
ch15f0	33.25	27.98	25.88
ch176f0	13.49	5.54	5.54
ch177f0	26.42	0	0
ch22f0	22.87	11.63	9.96
ch30f0	11.87	11.54	12.18
ch47f0	25.79	21.52	20.92
ch52f4	23.55	22.63	22.12
ch70f3	12.17	12.18	12.06
ch74f1	16.71	15.79	15.8
ch83f2	27.77	26.03	25.94
ch8f0	14.56	11.93	11.6
contains_4_ones	59.1	58.49	58.68
greater_than	15.89	16	16.03
interval1	32.95	34.28	33.92
interval2	44.03	44.35	44.43
kdd1	0.32	0.32	0.32
kdd10	17.73	17.52	17.96
kdd2	2.24	2.76	2.76
kdd3	1.6	1.28	1.28
kdd4	0	0	0
kdd5	11.2	10.52	10.46
kdd6	2.48	2.48	2.48
kdd7	19.08	20.69	21.81
kdd8	10.99	6.35	6.03
kdd9	21.54	13.79	13.63
majority_gate	34.85	36.24	35.54
modulus2	12.15	12.26	12.29
mux8	19.29	13.96	11.44
pal	15.96	16.67	16.67
pal_dbl_output	52.9	50.77	50.13
pal_output	58.84	58.98	59.39
parity	87.22	86.58	86.99
remaindr2	25.48	25.49	25.36
md_m1	2.42	2.38	2.38
md_m10	11.66	11.62	11.62
md_m25	26.37	25.59	25.59
md_m5	7.53	6.99	6.99
md_m50	42.75	42.72	43.01
md1	60.45	61.5	61.1
md2	61.52	62.25	61.84
md3	60.84	60.48	60.84
substr1	31.63	30.3	28.67
substr2	24.63	22.08	21.13
subtract1	48.15	41.42	41.43
subtract3	27.29	3.6	3.6
Average over all Functions	26.406875	23.22375	23.00625

Table 3: C4.5 trials with threshold size 0, varying the number of trees

Function Name	C4.5 1 tree Thresh=Wind.=0	C4.5 10 trees Thresh=Wind.=0	C4.5 100 trees Thresh=Wind.=0
add0	15.92	16.26	16.58
add2	35.4	27.4	26.32
add4	10	3.6	3.6
ch15f0	30.8	27.98	25.88
ch176f0	6.38	5.54	5.54
ch177f0	5.24	0	0
ch22f0	15.11	11.63	9.96
ch30f0	12.04	11.54	12.18
ch47f0	22.62	21.52	20.92
ch52f4	23.61	22.63	22.12
ch70f3	11.93	12.18	12.06
ch74f1	16.52	15.79	15.8
ch83f2	25.92	26.03	25.94
ch8f0	13.05	11.93	11.6
contains_4_ones	58.54	58.49	58.68
greater_than	15.95	16	16.03
interval1	33.53	34.28	33.92
interval2	45.06	44.35	44.43
kdd1	0.32	0.32	0.32
kdd10	16.4	17.52	17.96
kdd2	2.56	2.76	2.76
kdd3	1.28	1.28	1.28
kdd4	0	0	0
kdd5	11.2	10.52	10.46
kdd6	2.64	2.48	2.48
kdd7	19.87	20.69	21.81
kdd8	7.51	6.35	6.03
kdd9	16.16	13.79	13.63
majority_gate	35	36.24	35.54
modulus2	12.06	12.26	12.29
mux8	17.37	13.96	11.44
pal	16.35	16.67	16.67
pal_dbl_output	52.16	50.77	50.13
pal_output	58.1	58.98	59.39
parity	86.47	86.58	86.99
remaindr2	26.07	25.49	25.36
md_m1	2.38	2.38	2.38
md_m10	11.74	11.62	11.62
md_m25	25.59	25.59	25.59
md_m5	6.99	6.99	6.99
md_m50	42.65	42.72	43.01
rnd1	61.16	61.5	61.1
rnd2	61.63	62.25	61.84
rnd3	61.03	60.48	60.84
substr1	31.12	30.3	28.67
substr2	24.6	22.08	21.13
subtract1	45.09	41.42	41.43
subtract3	10	3.6	3.6
Average over all Functions	24.23166667	23.22375	23.00625

Table 4: C4.5 trials with window and threshold size 0, varying the number of trees

Function Name	C4.5 1 tree Threshold=-1	C4.5 10 trees Thresh=0, -1 256	C4.5 10 trees Threshold=1
add0	15.93	16.13	16.26
add2	44.78	28.69	27.42
add4	27.29	3.72	3.6
ch15f0	33.25	28.27	27.87
ch176f0	13.49	6.25	5.54
ch177f0	26.42	5.06	0
ch22f0	22.87	12.02	11.59
ch30f0	11.87	11.44	11.58
ch47f0	25.79	20.76	21.52
ch52f4	23.55	23.23	22.45
ch70f3	12.17	11.99	12.19
ch74f1	16.71	16.09	15.83
ch83f2	27.77	26.77	26.03
ch8f0	14.56	11.9	11.97
contains_4_ones	59.1	57.81	58.49
greater_than	15.89	15.55	16
interval1	32.95	32.94	34.1
interval2	44.03	45.33	44.2
kdd1	0.32	0.32	0.32
kdd10	17.73	17.29	17.4
kdd2	2.24	2.76	2.76
kdd3	1.6	1.28	1.28
kdd4	0	0	0
kdd5	11.2	11.39	10.52
kdd6	2.48	2.28	2.48
kdd7	19.08	20.13	20.7
kdd8	10.99	7.18	6.35
kdd9	21.54	14.24	13.79
majority_gate	34.85	35.75	36.24
modulus2	12.15	12.29	12.24
mux8	19.29	14.42	13.72
pal	15.96	16.77	16.67
pal_dbl_output	52.9	50.56	50.85
pal_output	58.84	59.23	58.92
parity	87.22	86.63	86.58
remaindr2	25.48	26.1	25.45
rnd_m1	2.42	2.14	2.38
rnd_m10	11.66	11.39	11.62
rnd_m25	26.37	25.57	25.59
rnd_m5	7.53	7.32	6.99
rnd_m50	42.75	42.16	42.72
rnd1	60.45	61.38	61.52
rnd2	61.52	61.2	62.22
rnd3	60.84	60.94	60.5
substr1	31.63	29.14	30.03
substr2	24.63	22.58	22.33
subtract1	48.15	42.05	41.44
subtract3	27.29	3.72	3.6
Average over all Functions	26.406875	23.37833333	23.2052083

Table 5: C4.5 trials with varying options

Function Name	C4.5 10 trees	C4.5 10 trees
	Threshold=0/grouping (-s)	Threshold=1/grouping (-s)
add0	16.26	16.26
add2	27.4	27.42
add4	3.6	3.6
ch15f0	27.98	27.87
ch176f0	5.54	5.54
ch177f0	0	0
ch22f0	11.63	11.59
ch30f0	11.54	11.58
ch47f0	21.52	21.52
ch52f4	22.63	22.45
ch70f3	12.18	12.19
ch74f1	15.79	15.83
ch83f2	26.03	26.03
ch8f0	11.93	11.97
contains_4_ones	58.49	58.49
greater_than	16	16
interval1	34.28	34.1
interval2	44.35	44.2
kdd1	0.32	0.32
kdd10	17.52	17.4
kdd2	2.76	2.76
kdd3	1.28	1.28
kdd4	0	0
kdd5	10.52	10.52
kdd6	2.48	2.48
kdd7	20.69	20.7
kdd8	6.35	6.35
kdd9	13.79	13.79
majority_gate	36.24	36.24
modulus2	12.26	12.24
mux8	13.96	13.72
pal	16.67	16.67
pal_dbl_output	50.77	50.85
pal_output	58.98	58.92
parity	86.58	86.58
remainder2	25.49	25.45
rnd_m1	2.38	2.38
rnd_m10	11.62	11.62
rnd_m25	25.59	25.59
rnd_m5	6.99	6.99
rnd_m50	42.72	42.72
rnd1	61.5	61.52
rnd2	62.25	62.22
rnd3	60.48	60.5
substr1	30.3	30.03
substr2	22.08	22.33
subtract1	41.42	41.44
subtract3	3.6	3.6
Average over all Functions	23.22375	23.20520833

Table 6: C4.5 trials with grouping options

any significant benefit going from 10 trees to 100. In fact, we have no reason to believe that there will be any significant difference between 10 trees and 1000. Thus our best set for which we will test against Pattern Theory will be with all the options set at default except the weight (threshold) will be 0 (-m 0) and the number of trees will be 10 (-t 10). We will use -m 0 over -m 1 because this is the lowest setting we can have that corresponds to not having any noise in the data.

5 Experimental Results

Now that we have the best options possible for C4.5 over our benchmark set of functions, we can test it with honesty against Pattern Theory. This section refers to the learning curves for every function tested. The curves themselves are shown in Appendix A. The sets are displayed with C4.5 first on a given function, then Pattern Theory. For a given graph, the Y-axis is the number of errors and the X-axis is the number of training samples. Each graph includes the maximum, minimum, and average error. The experiments stopped if the maximum error reached 0 (i.e., for all 10 runs, there were no errors). Thus, there would not be any data points beyond that particular sample size. The chance line, represented by dashes, is the error expected if we were to randomly guess on the remaining cases. We would expect to get half of them right and half wrong since there are only two outcomes. On the graphs for FLASH (Pattern Theory), there are additional points plotted corresponding to the calculated DFC and the number of "don't cares" for a given sampling. We can also see that functions that are highly patterned have a low DFC while more complicated patterns have a higher DFC. Moreover, the random functions have a very high DFC.

Earlier, we gave some background about how function decomposition works and thus how FLASH works. What has not been described is the actual search procedure that FLASH uses in order to select a partition. First, the same strategy was used for every experiment. Essentially, it is a two-ply look ahead on all possible partitions. The calculated DFC is used to continue selecting partitions until they no longer decompose. The actual strategy itself is given in Appendix B. The name of the decomposition plan is dni0e300.

6 Analysis

If we examine C4.5's performance as a whole, its ability ranges from extremely good to extremely poor. C4.5's performance was excellent for the Boolean Expression functions; it had a respectable performance for PAL, MUX8, MODULUS2, and GREATER_THAN; it has a hard time with the other string functions, and it is poor at learning the other PAL functions. C4.5 is especially poor at PARITY. Its performance on the character functions are mixed. Some it learns very well and others, the performance is fair. A few anomalies were the fact that C4.5 performed very well on ADD4 and ADD0 but poorly on ADD2. It was even more bizarre to see excellent performance on SUBTRACTION3 and very poor performance on SUBTRACTION1.

Comparing C4.5 and FLASH (Pattern Theory/Function Decomposition), C4.5 beats FLASH for only two functions: KDD2 and KDD3. It is equal or slightly better for the two character functions: CH52F4 and CH83F2. For all of the other functions however, FLASH outperforms C4.5. In some cases, the performance margin is substantial. The notable cases are: ADD2, ADD4, CONTAINS_4_ONES, KDD7, KDD9, KDD10, MAJORITY_GATE, PARITY, SUBTRACTION1, and SUBTRACTION3. Of course, we are not concerned with comparing performance on different random functions. Their purpose is to measure consistency and normal behavior. It would be unusual for any method to be significantly better in some random function than another.

The other functions were not mentioned here because some comparisons might be construed as unobjective. Although different performance is seen in some cases, in general, we see equal performance or FLASH performing better. The above functions were mentioned specifically because of the vast differences between the two programs.

In general what we see in C4.5 is that it is unequipped to handle "XOR" type relations. The inherent problem is its inability to deal with replication in such disjunctive concepts as: (A and B) or (C and D). This is as expected [6]. It would appear that C4.5 is unable to effectively learn functions that have an "XOR" or lend themselves to "XOR." FLASH, on the other hand, has no restrictions in this area. There are still some problems with functions that have deep replication that prevent FLASH from completely learning such a function unless all of the samples are given. However, its performance does not degrade beyond C4.5.

C4.5 holds its own for the Boolean functions that do not involve "XOR." It also had a respectable performance for some of the character functions. However, the best domain for C4.5 is the class of Boolean Expressions. In the other areas, it does not stand up to FLASH.

In Table 7, we list the average errors for all the functions, similar to the previous section. Here, however, we show C4.5's best with FLASH. One can see from the total average error that FLASH is outperforming C4.5 as a robust pattern finder in this domain of binary variables and noise free data. This table also shows off to the side, how the average error changes after successively removing functions that C4.5 is unable to handle. Once they are all removed, their respective performances are nearly equal.

It is noted to the reader that although the Table 7 provides a nice compact comparison, it is not 100% reliable. There are a few functions in which the average error does not correspond to performance. They are anomalous and are few in number. They are notably KDD2, Subtraction3, and ADD4. For example in KDD2, C4.5's average error is 2.76 versus FLASH's 2.4. However, C4.5 learns the function in 125 samples and FLASH learns the function in 150 samples. On the other hand in Subtraction3, C4.5's average is 3.6 versus FLASH's 0. The two performances appear similar. But, C4.5 learns the function in 150 samples and FLASH learns the function in only 25 samples. For a comprehensive analysis, the reader is again referred to the individual learning curves in Appendix A.

Another attempt is made to summarize all of the data from the graphs in Appendix A shown in Figure 3. Here, we show the number of functions learned versus the number of samples for FLASH and C4.5. From the figure, we can see a clear performance distinction between the two methods.

7 Conclusion and Summary

In conclusion, FLASH (Pattern Theory) was shown to be a more robust pattern finder than C4.5 for our limited domain of binary variables and noise free data. Again, we emphasize the point that Pattern Theory can be extended to discrete and continuous valued variables demonstrating its flexibility. C4.5 held its own in the Boolean Expression domain and some of the images, however, its performance was lacking in comparison in the other domains. Specifically, C4.5 fails to learn concepts with implicit "XOR" representations or functions that have duplication in their subtrees.

Pattern Theory has been demonstrated a robust, effective inductive learning technique comparable to the best. The experimental results show its learning ability relative to chance and C4.5. Furthermore, as displayed by our graphs, there is a correlation between a function that is highly "patterned" and a function that has a low DFC.

8 Future Work

Some future directions in this area are to continue testing more functions like those in our experiments. In fact, a few of the functions tested were added after most of the experiments were performed (the monk problems and the "deep functions") and were not included in the first 7 tables. New functions are constantly being tested, but we had to wrap up the discussion at some point. However, their graphs were included in the Appendix A for study. In addition, it is planned to increase the number of variables to as many as 30 in the immediate future. We are also looking into adapting the current program to handle discrete and continuous variables. Furthermore, we ultimately plan to incorporate methods of handling noise. Moreover, we are looking for ways to increase the speed by limiting the exploration of the partition search space.

We have a working theoretical result of applying function decomposition to continuous variables and the searching ability is getting better. At this point, the real limitation is the number of variables and noise. Since function decomposition involves an exponential search space, the only hope is using some method to prune the branches of the tree. At the rate the work is going, it is very possible that at the time of this printing, we will be able to handle up to 100 variables with the same accuracy.

Noise, on the other hand, is a more difficult problem. At present, we have no formal theoretical basis for dealing with it. It is, however, a personal interest of the author and the hope is to perform some quality research in this area.

Function Name	C4.5 10 trees Threshold = 0	Flash dni0e300			
add0	16.26	10.38			
add2	27.4	5.24			
add4	3.6	0			
ch15f0	27.98	19.595			
ch176f0	5.54	0.16			
ch177f0	0	0			
ch22f0	11.63	7			
ch30f0	11.54	10.29			
ch47f0	21.52	16.89			
ch52f4	22.63	27.74			
ch70f3	12.18	12.04			
ch74f1	15.79	15.85			
ch83f2	26.03	27.885			
ch8f0	11.93	11.7			
contains_4_ones	58.49	24.49			
greater_than	16	9.78			
interval1	34.28	33.585			
interval2	44.35	35.94			
kdd1	0.32	0			
kdd10	17.52	8.18			
kdd2	2.76	2.4			
kdd3	1.28	2.72			
kdd4	0	0			
kdd5	10.52	11.11			
kdd6	2.48	3.72			
kdd7	20.69	10.53			
kdd8	6.35	0			
kdd9	13.79	6.55			
majority_gate	36.24	18.74			
modulus2	12.26	13.4			
mux8	13.96	13.04			
pal	16.67	9.9			
pal_dbl_output	50.77	38.94			
pal_output	58.98	58.79			
parity	86.58	10.45			
remainder2	25.49	25.22			
md_m1	2.38	2.31	Average	C4.5	Flash
md_m10	11.62	13.045	All Functions	23.22375	17.58521
md_m25	25.59	25.61	Without Parity	21.87574	17.73702
md_m5	6.99	7.815	Without KDD's Having XOR	22.51136	18.61159
md_m50	42.72	42.685	Without Majority Gate	22.19209	18.6086
md1	61.5	59.125	Without Contains 4 1's	21.32786	18.46857
md2	62.25	60.055	Without Subtraction	21.26875	18.7865
md3	60.48	59.865	Without ADD's	21.71595	19.88757
substr1	30.3	24.105			
substr2	22.08	23			
subtract1	41.42	24.22			
subtract3	3.6	0			
Average over all fns	23.22375	17.58521			

Table 7: C4.5's best options with FLASH's best

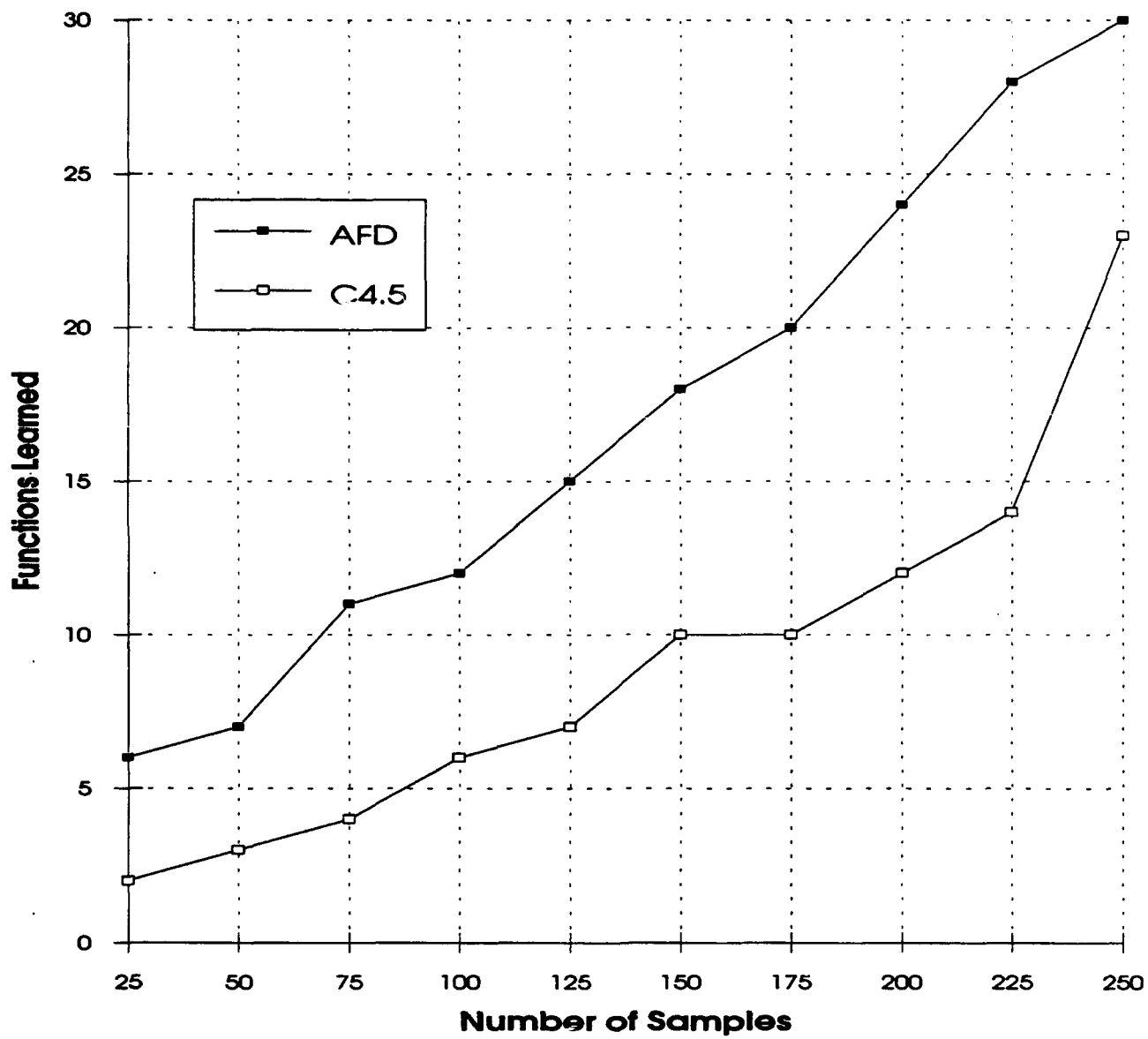


Figure 3: Generalisation Comparison

References

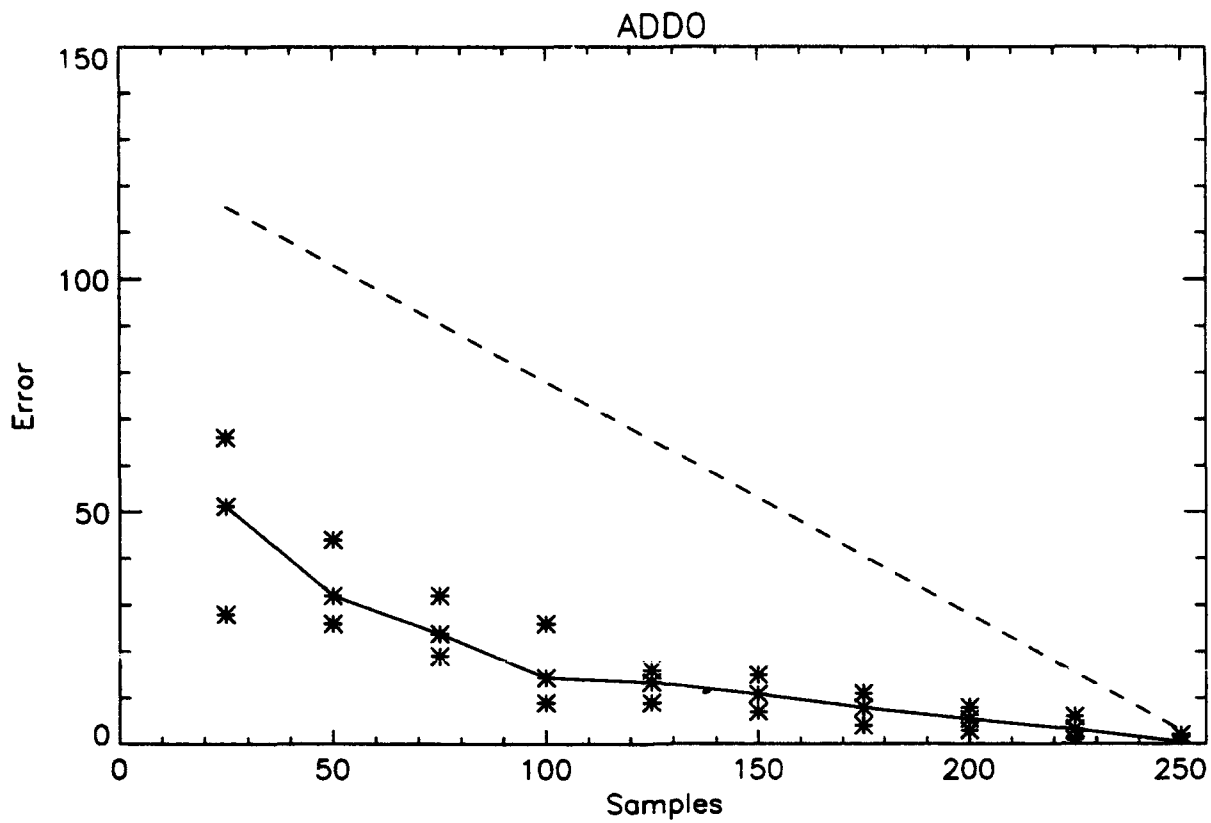
- [1] Robert L. Ashenurst. The decomposition of switching functions. In *Proceedings of the International Symposium on the Theory of Switching*, April 1957.
- [2] Mark L. Axtell, Timothy D. Ross, and Michael J. Noviskey. Pattern theory in algorithm design. In *NAECON Proceedings*. IEEE and AIAA, May 1993.
- [3] Jeffrey A. Goldman. Pattern theoretic knowledge discovery. Technical Report WL-TR-94-1093, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1994. work in progress.
- [4] Jeffrey A. Goldman. Pattern theoretic knowledge discovery. In *6th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, November 1994.
- [5] L. Hyafil and R.L. Rivest. Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters*, 5(1):15-17, 1976.
- [6] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Maching Learning*, 5:71-99, 1990.
- [7] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, Palo Alto, California, 1993.
- [8] Timothy D. Ross, Michael J. Noviskey, Timothy N. Taylor, and David A. Gadd. Pattern theory: An engineering paradigm for algorithm design. Final Technical Report WL-TR-91-1060, Wright Laboratory, USAF, WL/AART, WPAFB, OH 45433-6543, August 1991.
- [9] S. B. Thrun and et. al. The monk's problems - a performance comparison of different learning algorithms. Technical report, Carnegie Mellon University, December 1991.

A Individual Learning Curves of Each Function for C4.5 and FLASH

This section is a set of graphs described in the report. Every graph has the name of the function being tested at the top, the number of errors as the y -axis, and the number of samples as the x -axis.

The tests on the individual functions were as follows. First, each method was given a random set of data to train on ranging from 25 to 250 out of a total of 256 possible cases. Once the method was trained, the entire 256 cases were tested and the number of differences were recorded as errors. This procedure was repeated 10 times for a given sample training size in intervals of 25 yielding a maximum, minimum, and average number of errors for each. Thus, the total number of runs for each function was 100 of varying sample size. None of the learning was incremental. All of the runs were independent.

The chance line was calculated as follows. For a given sample size, assume we simply mimic the data we are given and then randomly guess the remaining unknown elements. This creates a static learning line which represents learning by chance. For a given function, the graph for C4.5 is displayed first followed by FLASH (function decomposition). The FLASH graphs also have some additional information plotted: the average DFC, and the number of "don't cares" or unknowns. The DFC is calculated by the function realized for each sample size. Since there are ten trials at each sample size, an average DFC is computed.



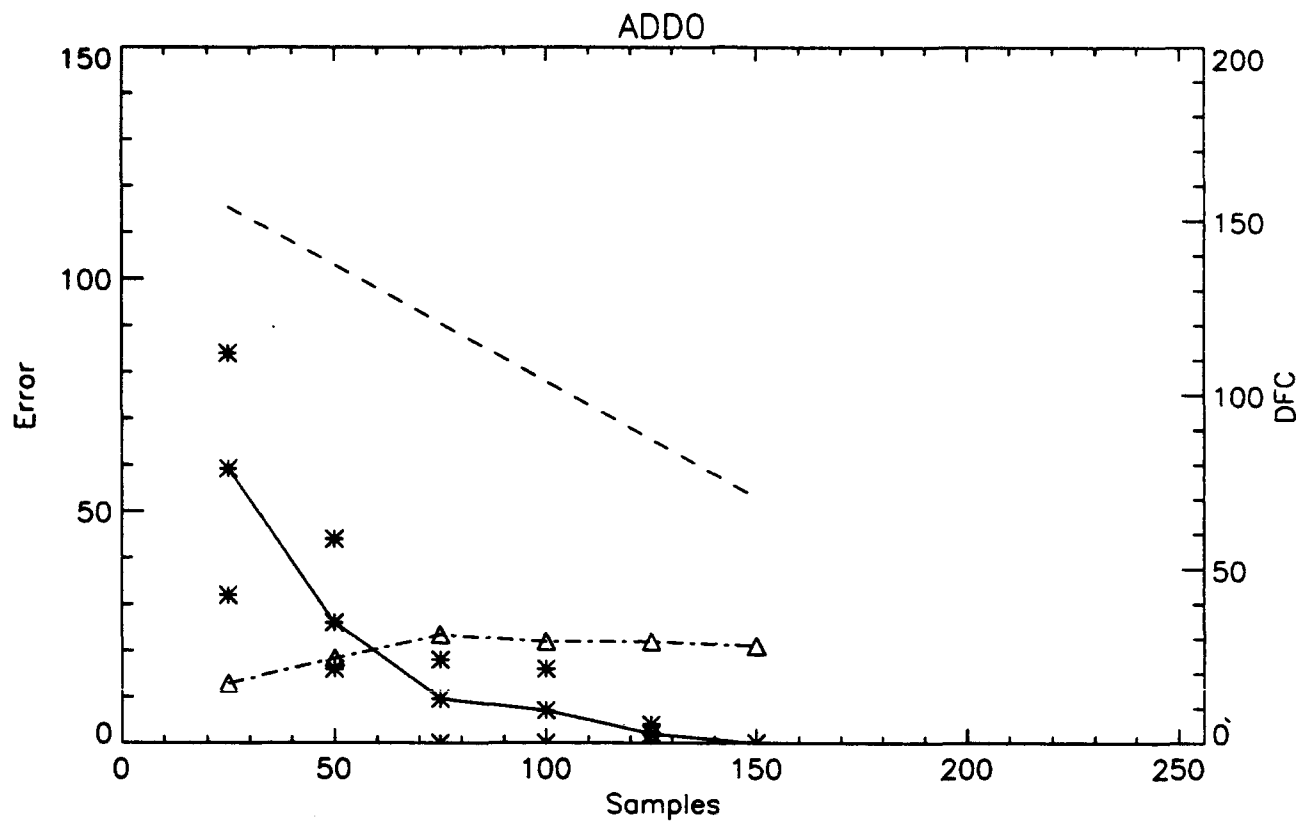
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

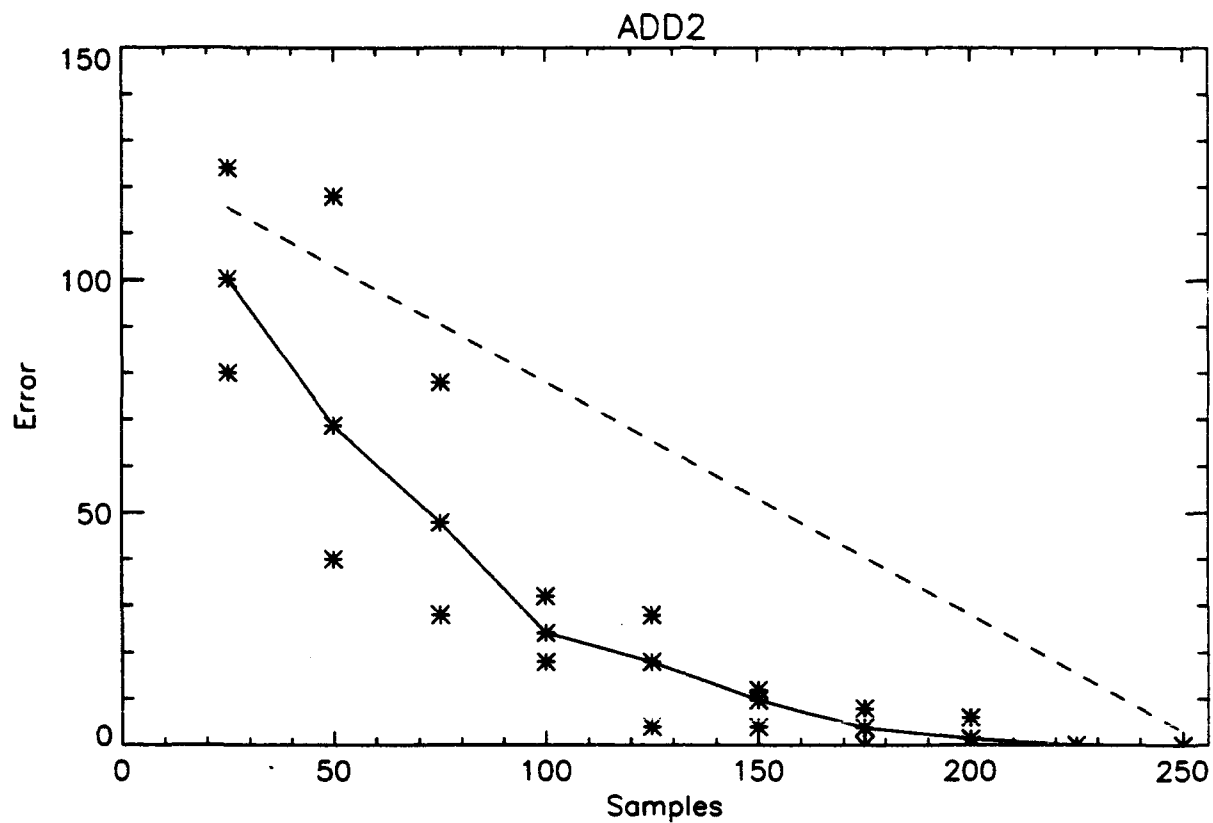
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



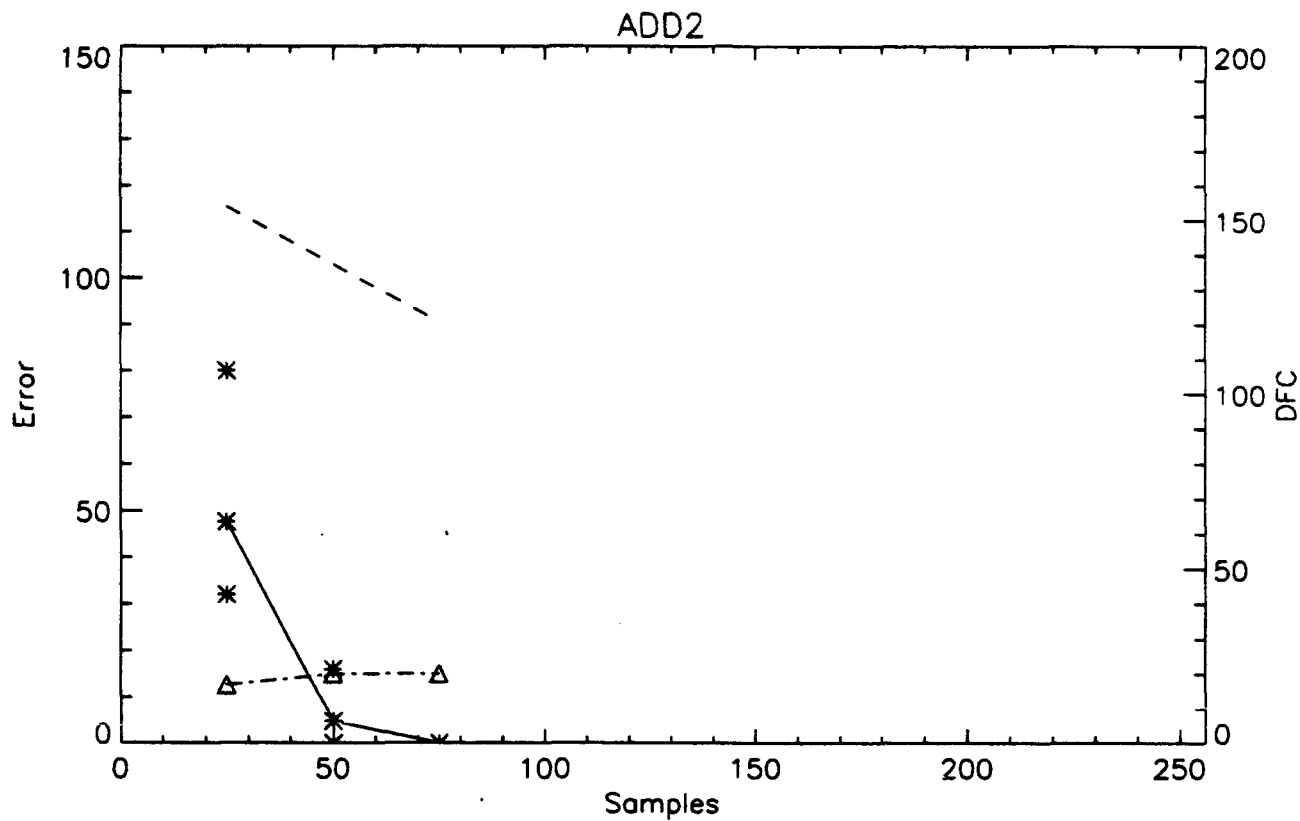
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

—*— Avg Error



- Chance

- * Max error

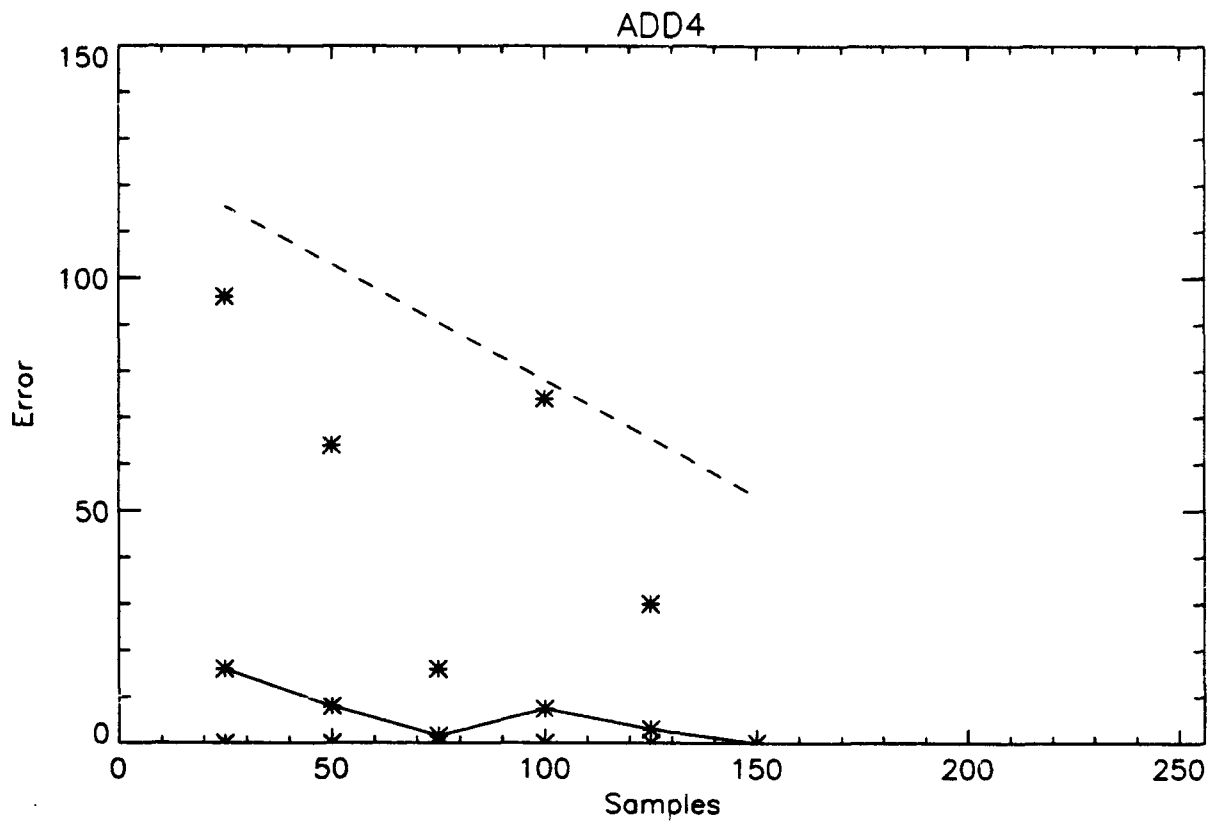
- * Min error

- *— Avg error

-◇..... Don't cares

- △--- Avg DFC

FLASH dni0e300



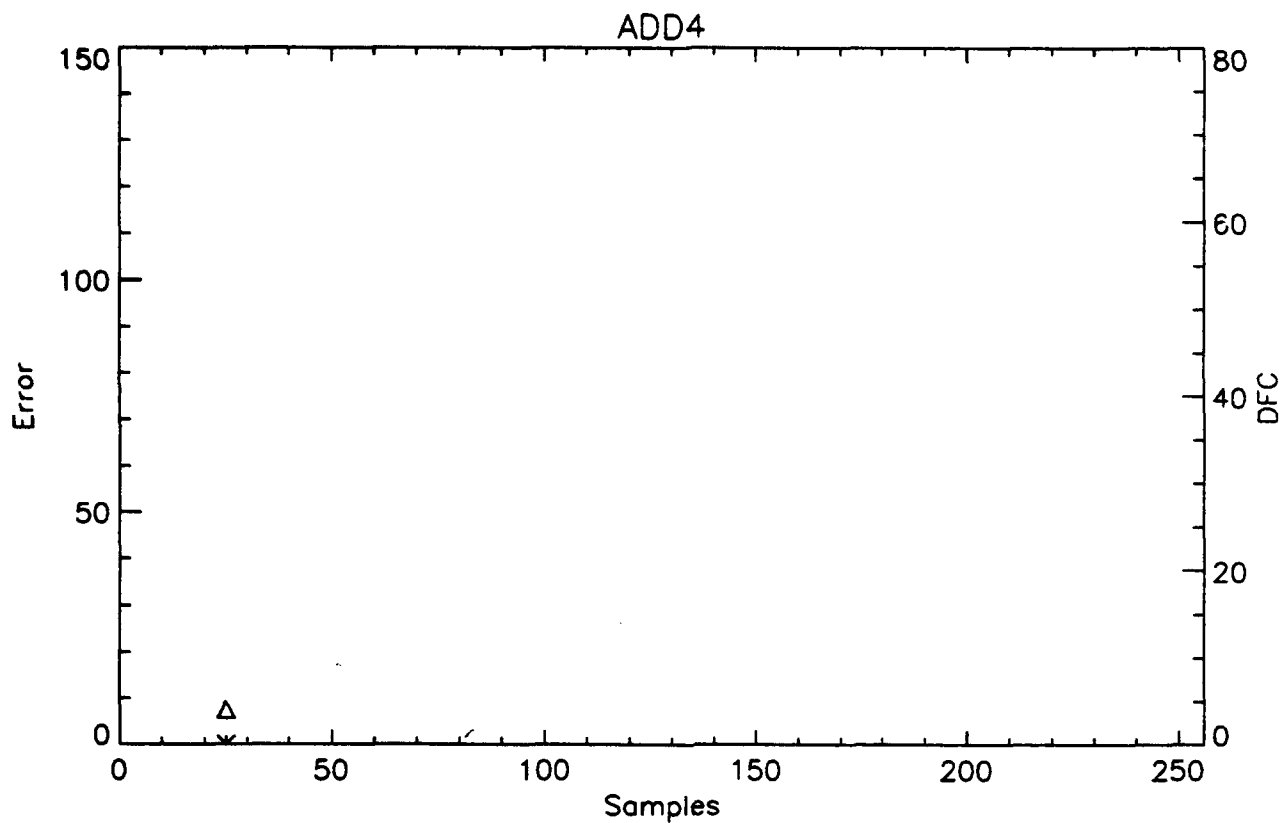
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

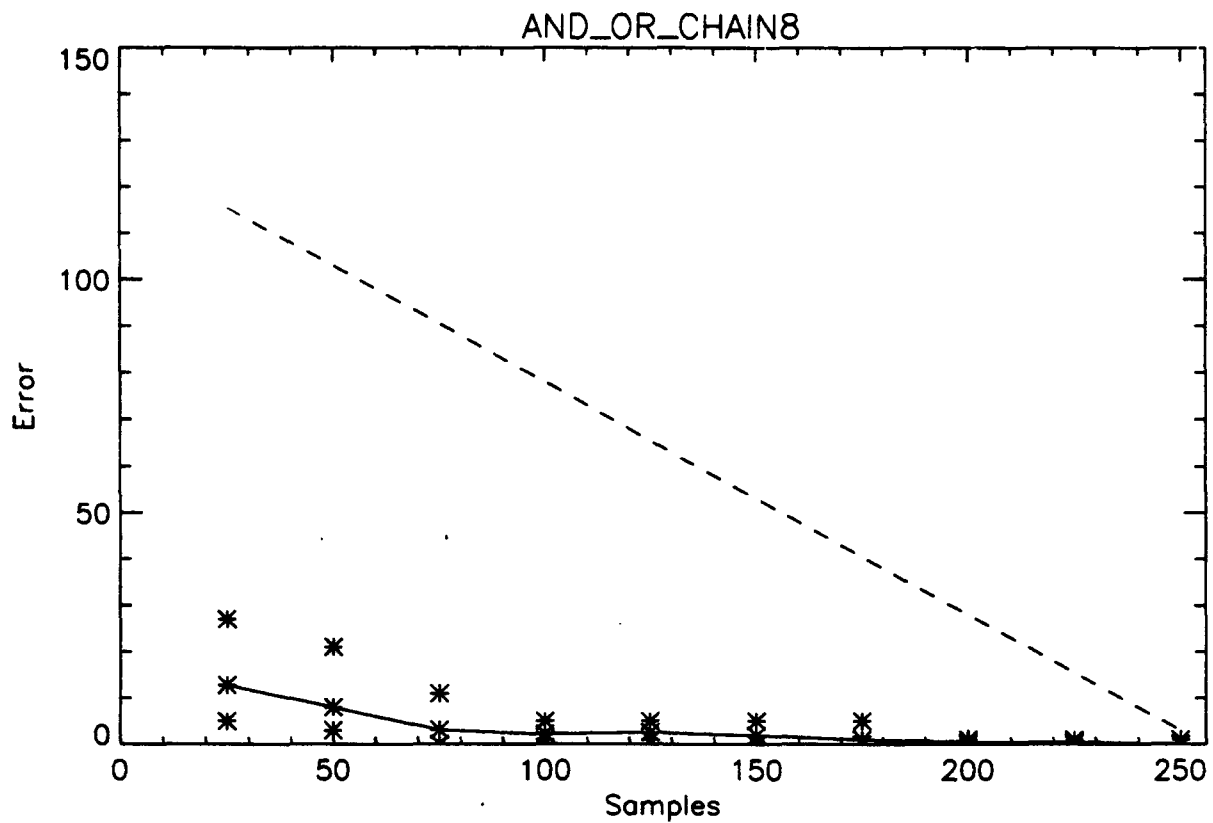
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



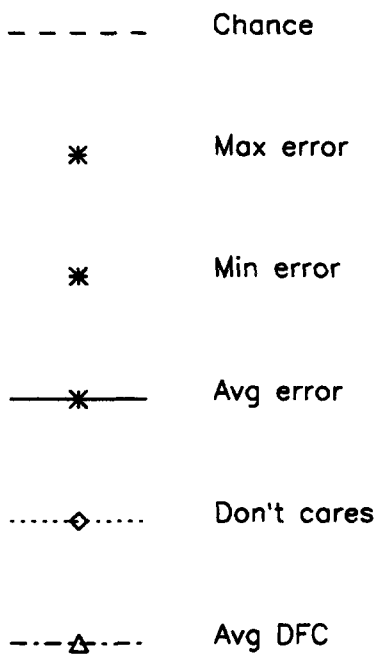
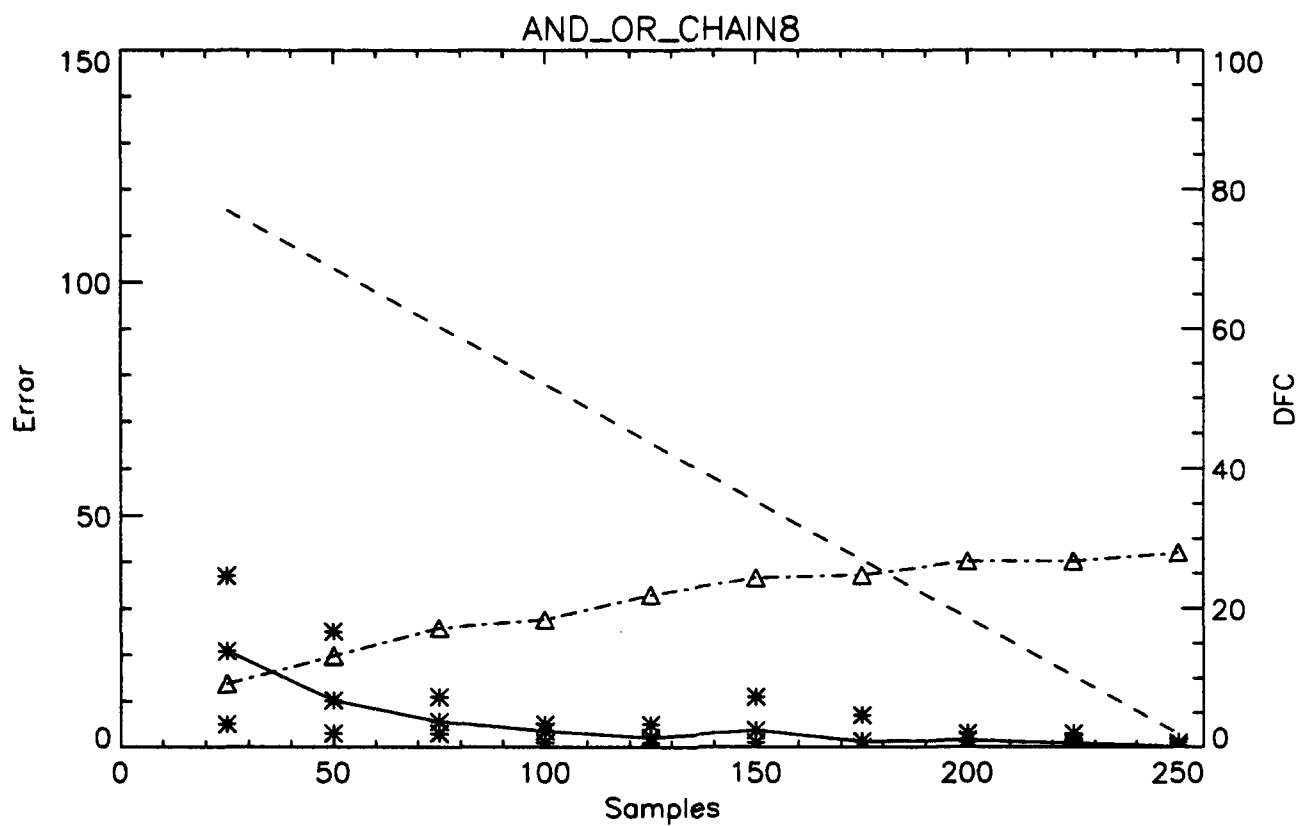
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

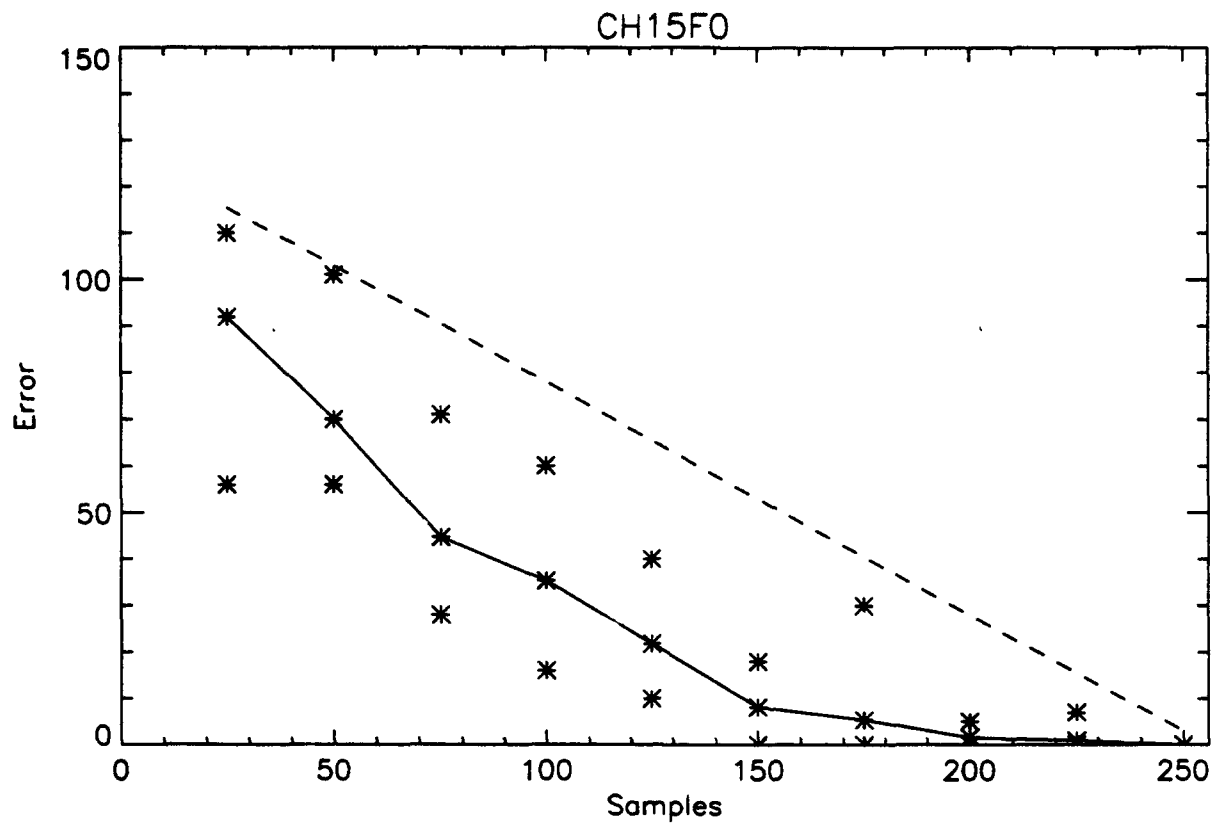
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



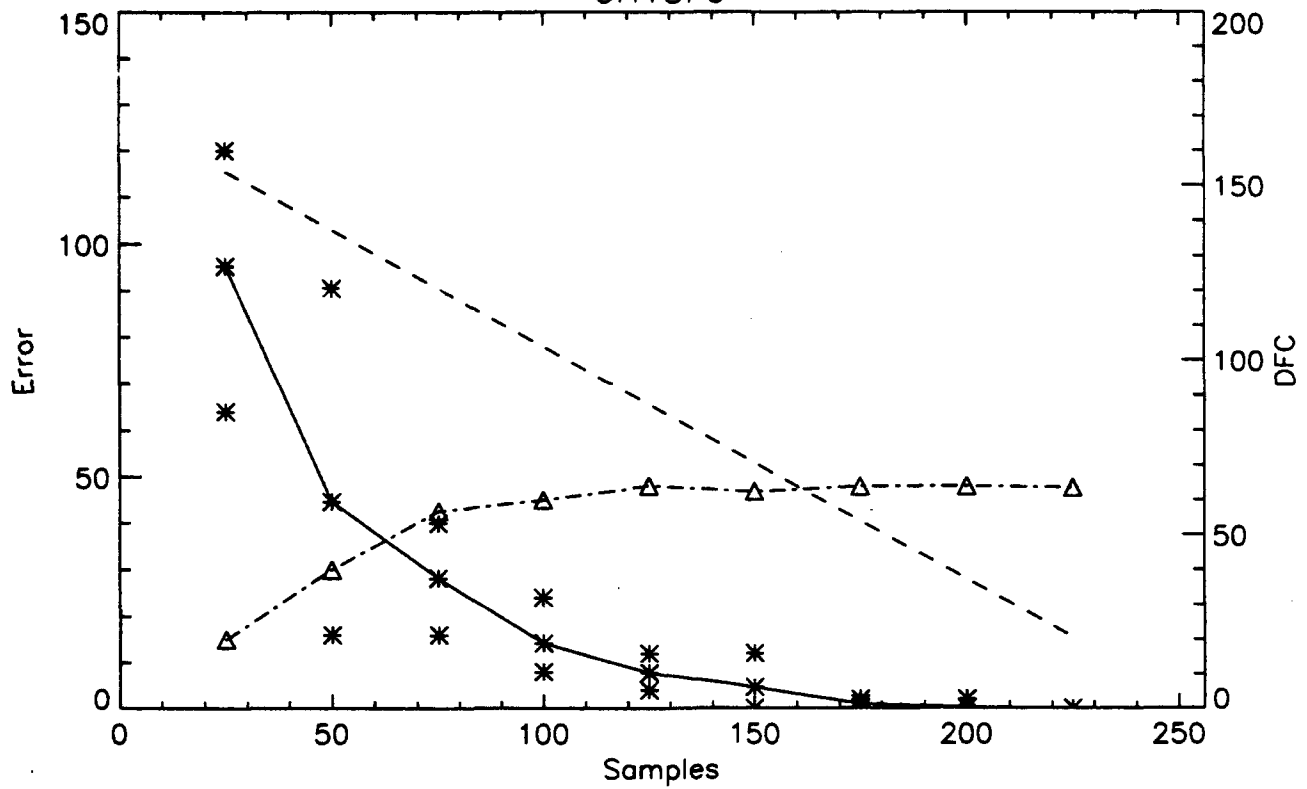
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

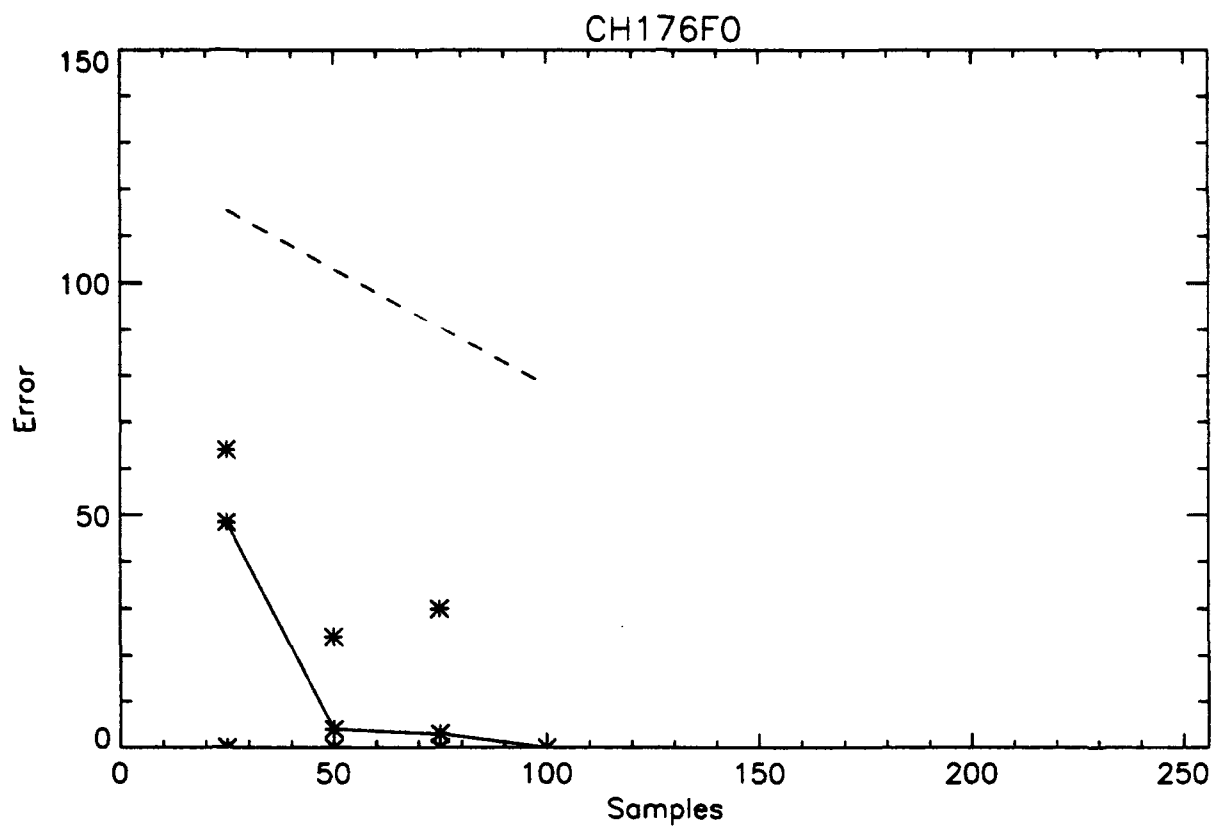
—*— Avg Error

CH15F0



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



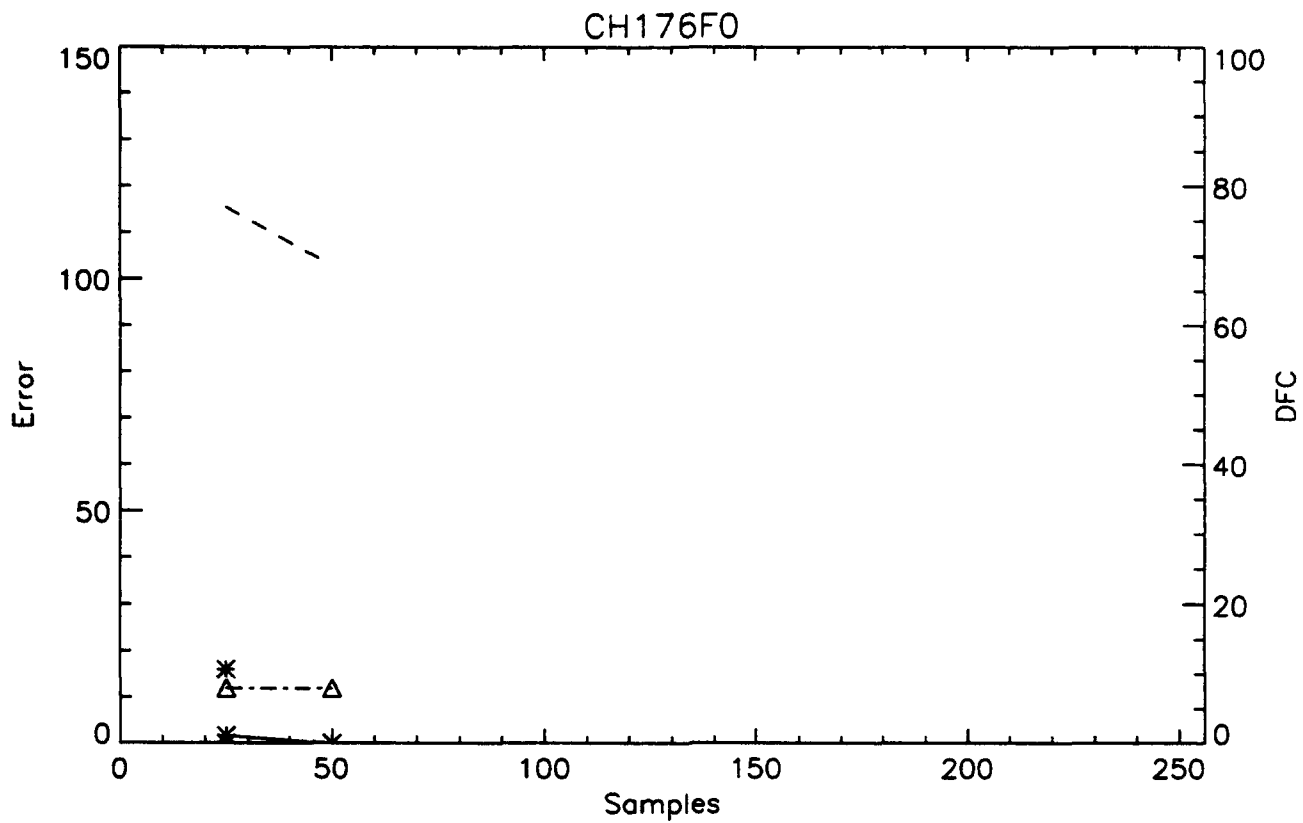
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

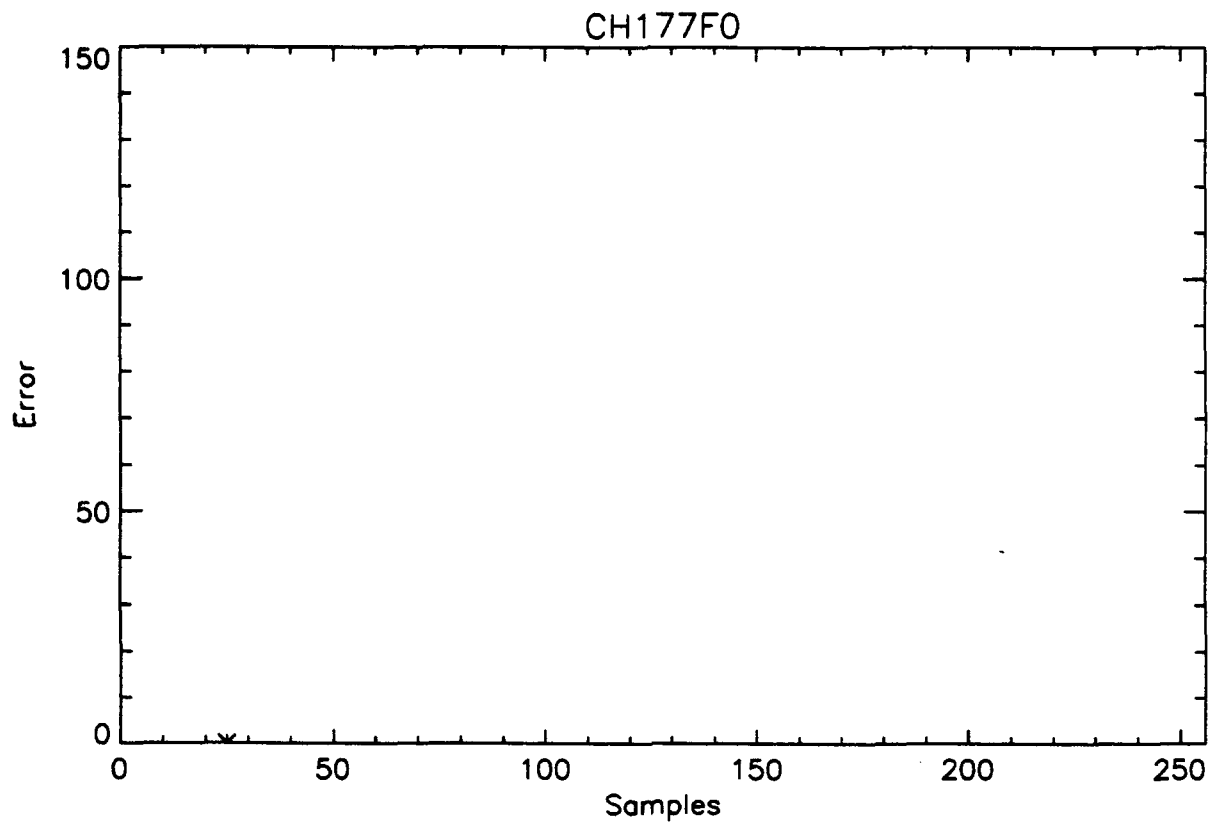
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-△-.- Avg DFC

FLASH dni0e300



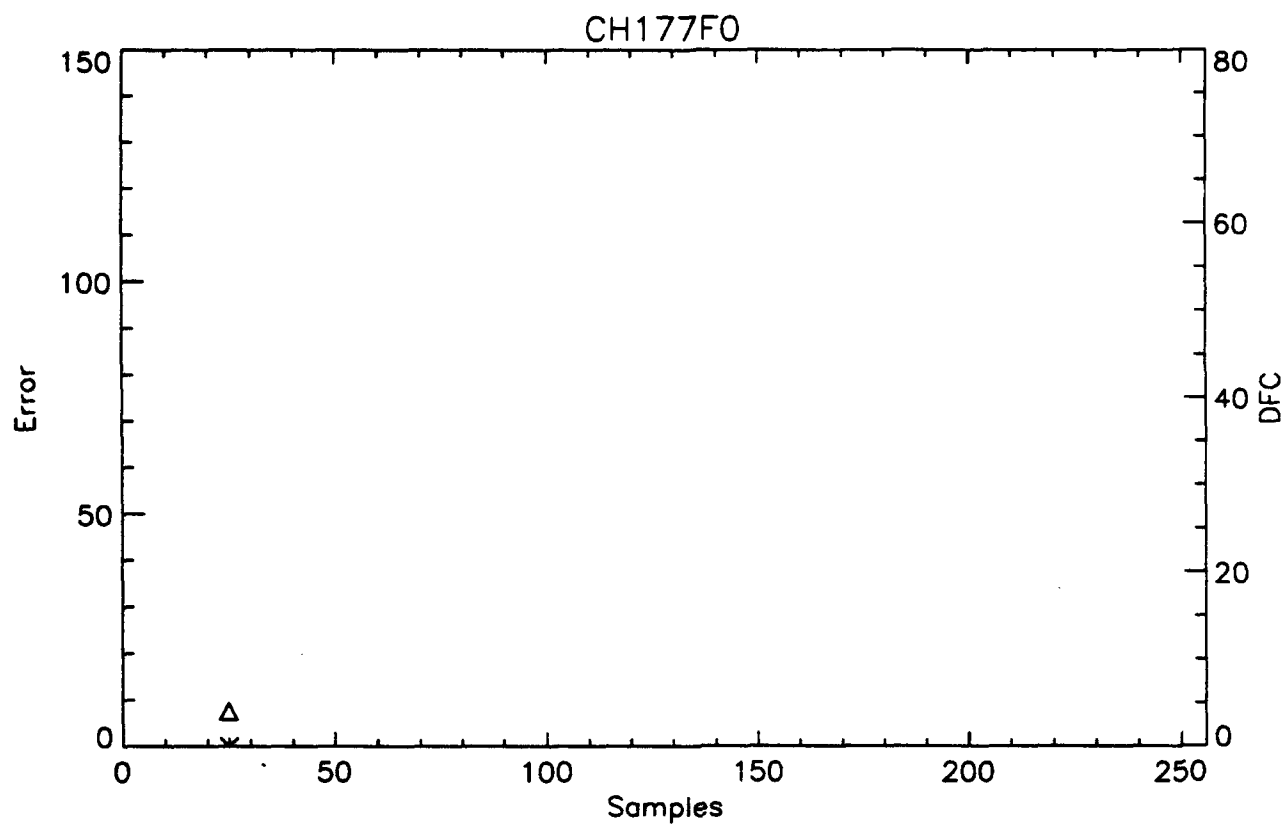
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

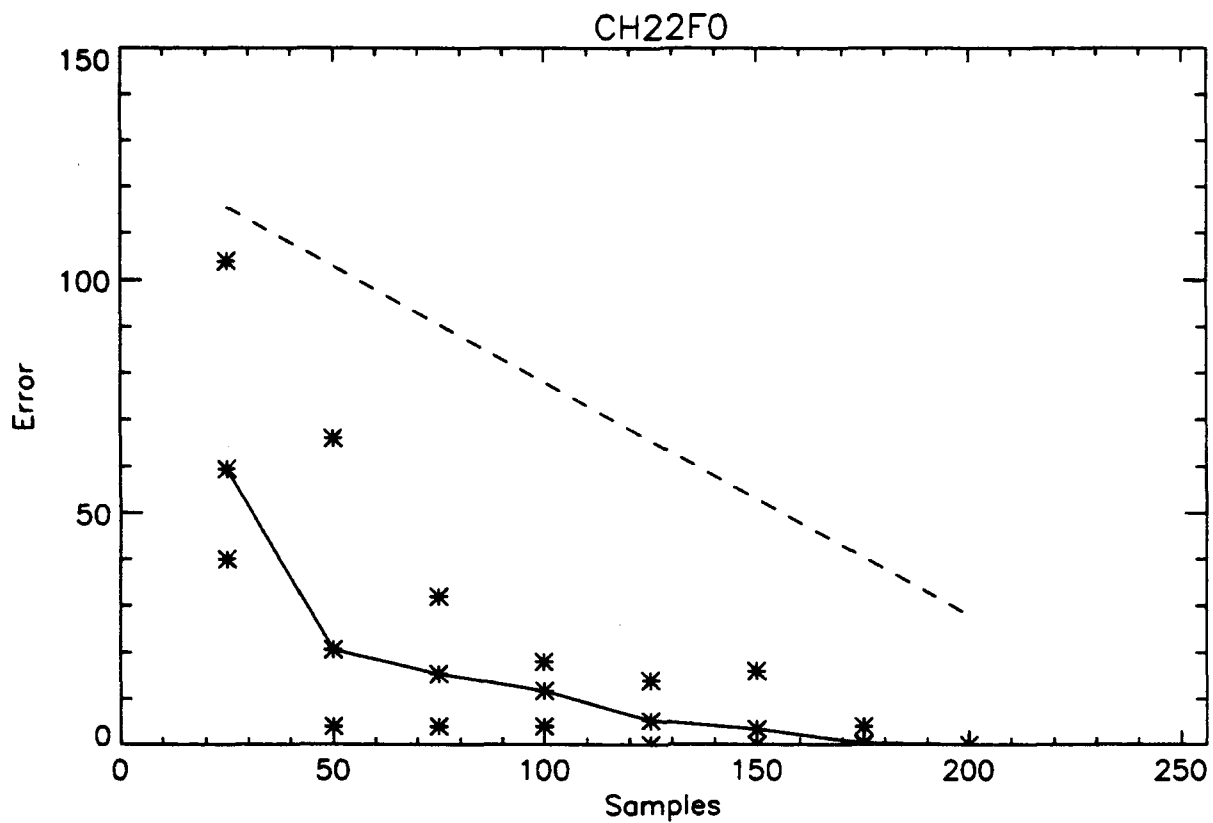
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



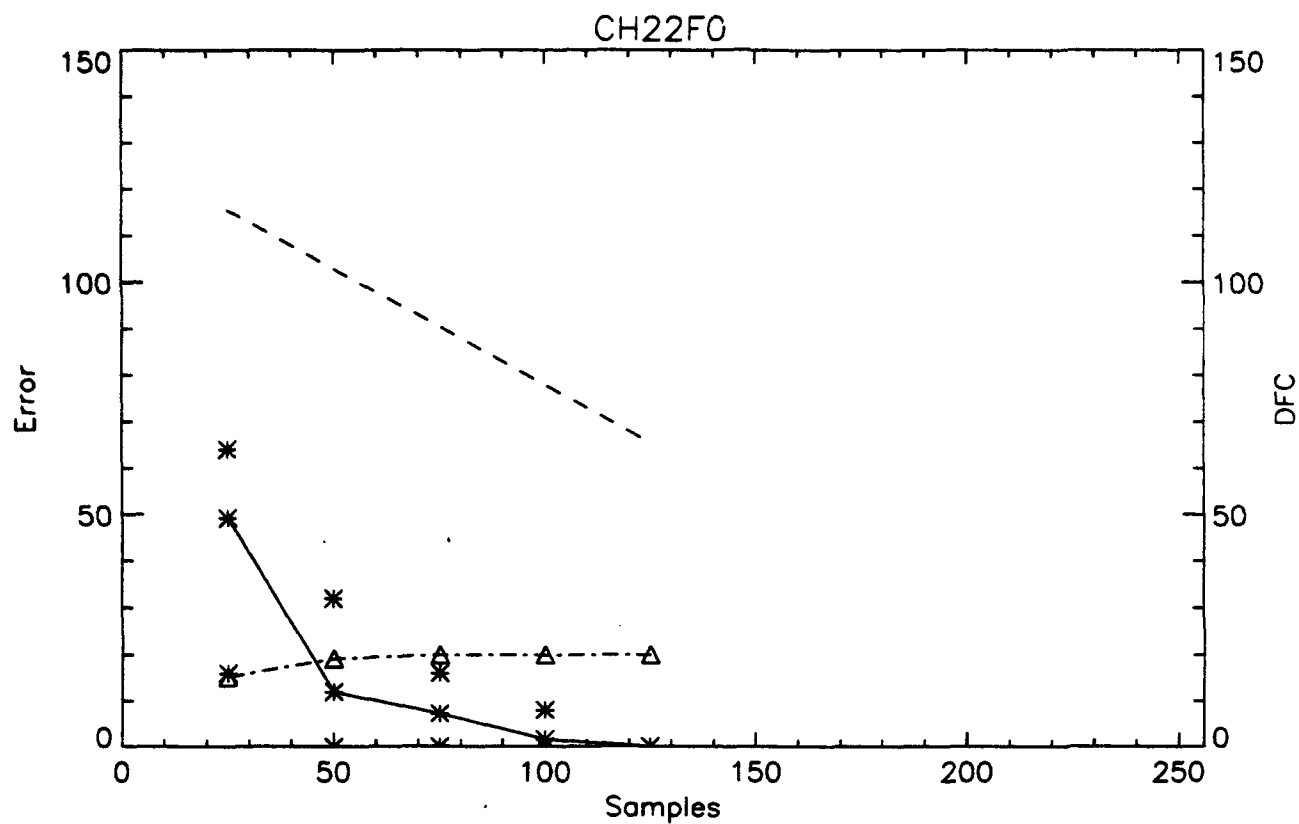
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

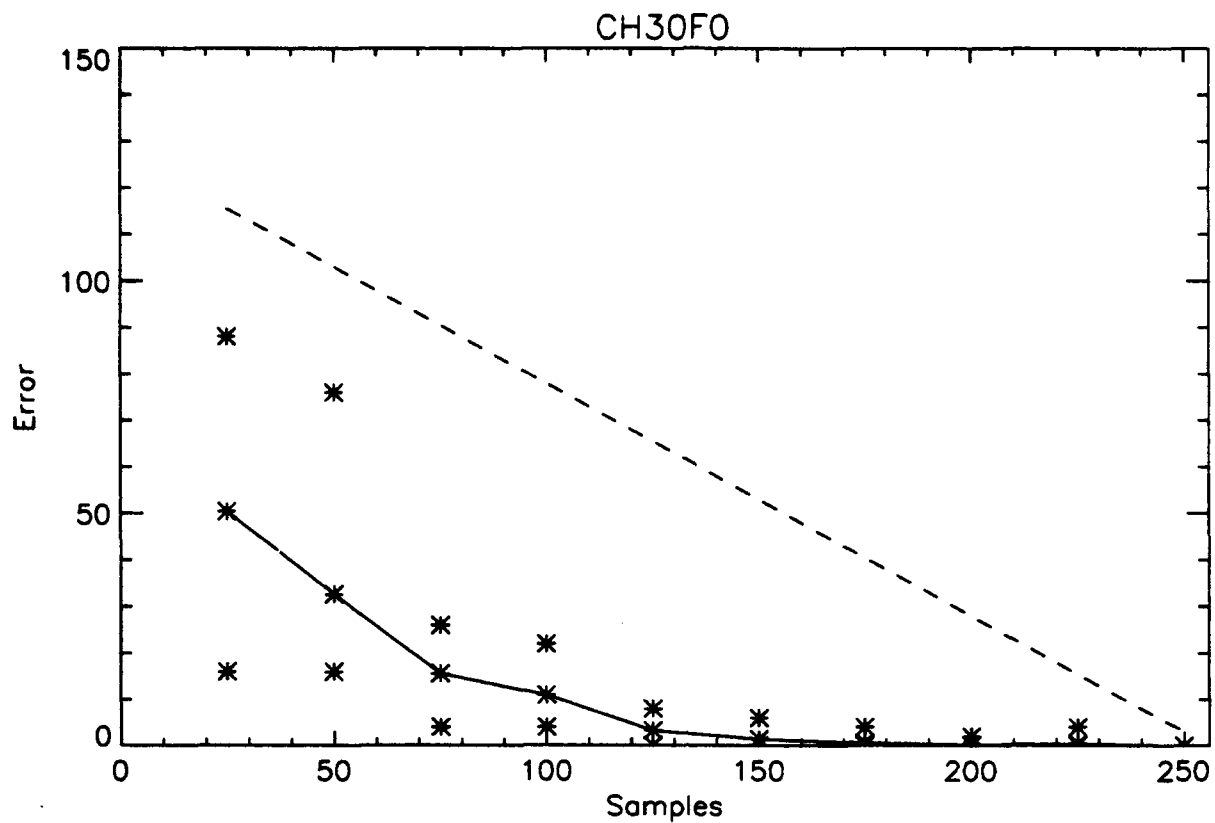
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



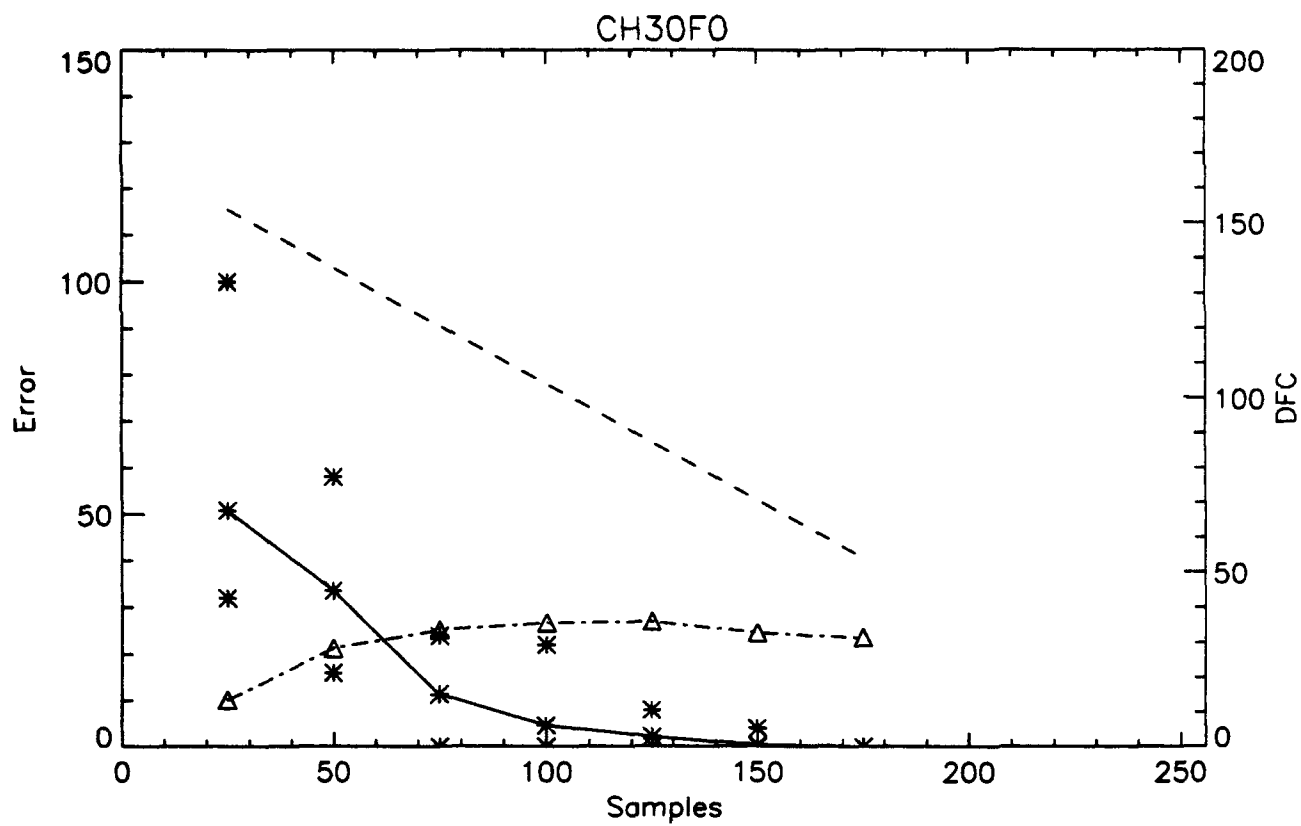
--- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

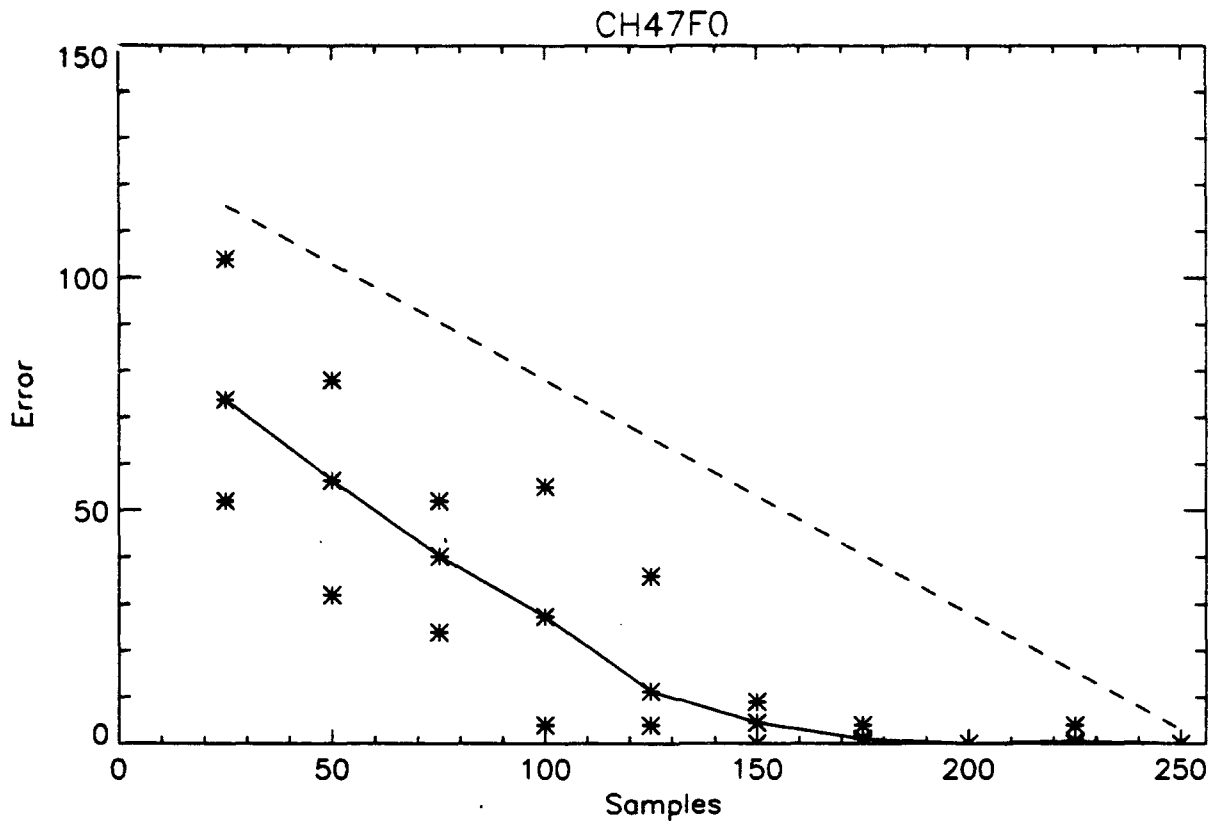
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



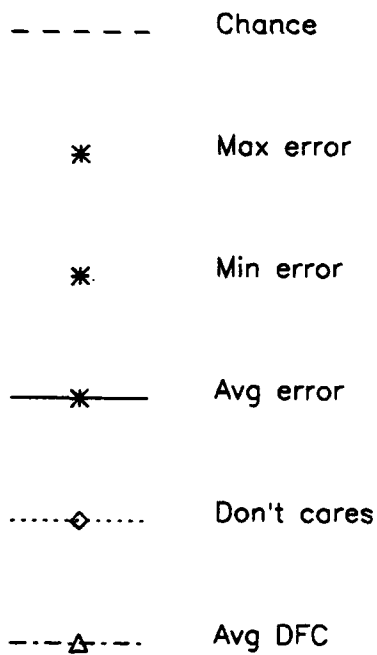
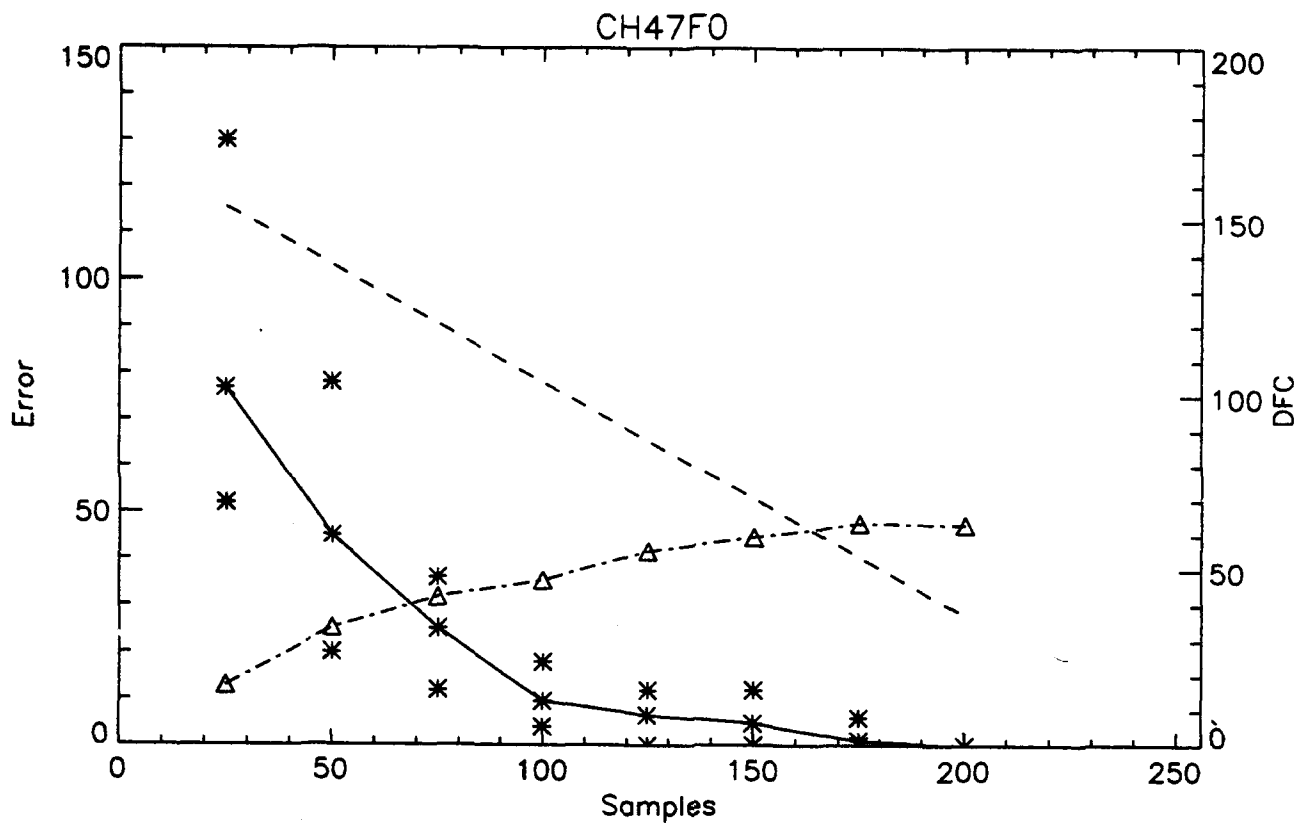
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

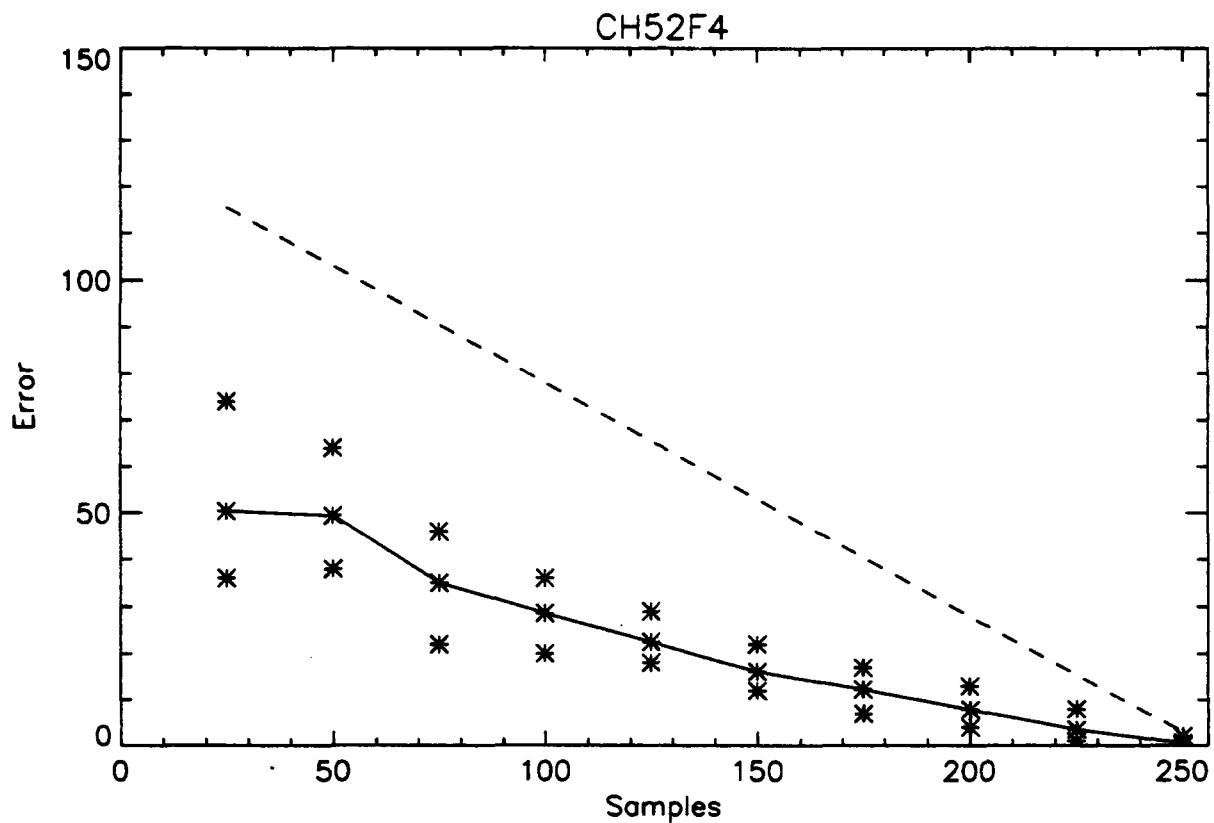
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



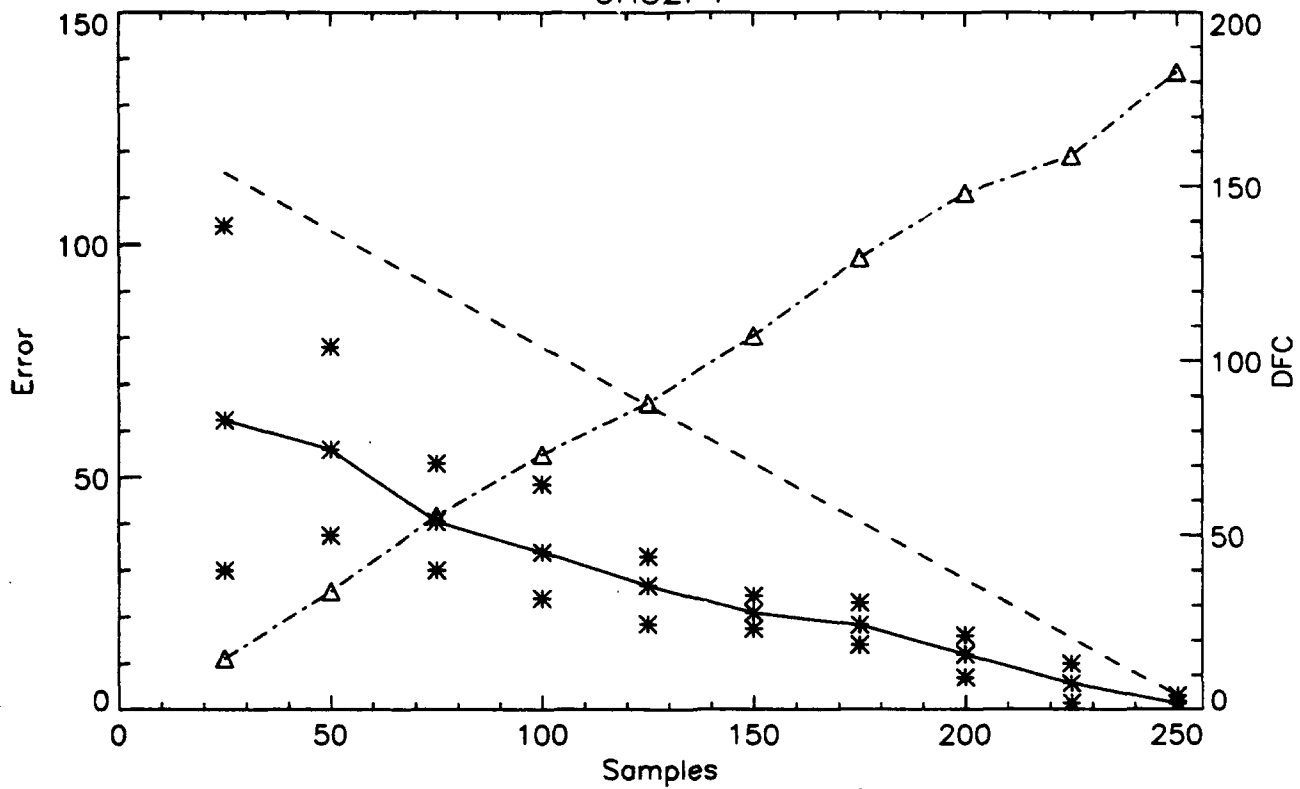
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

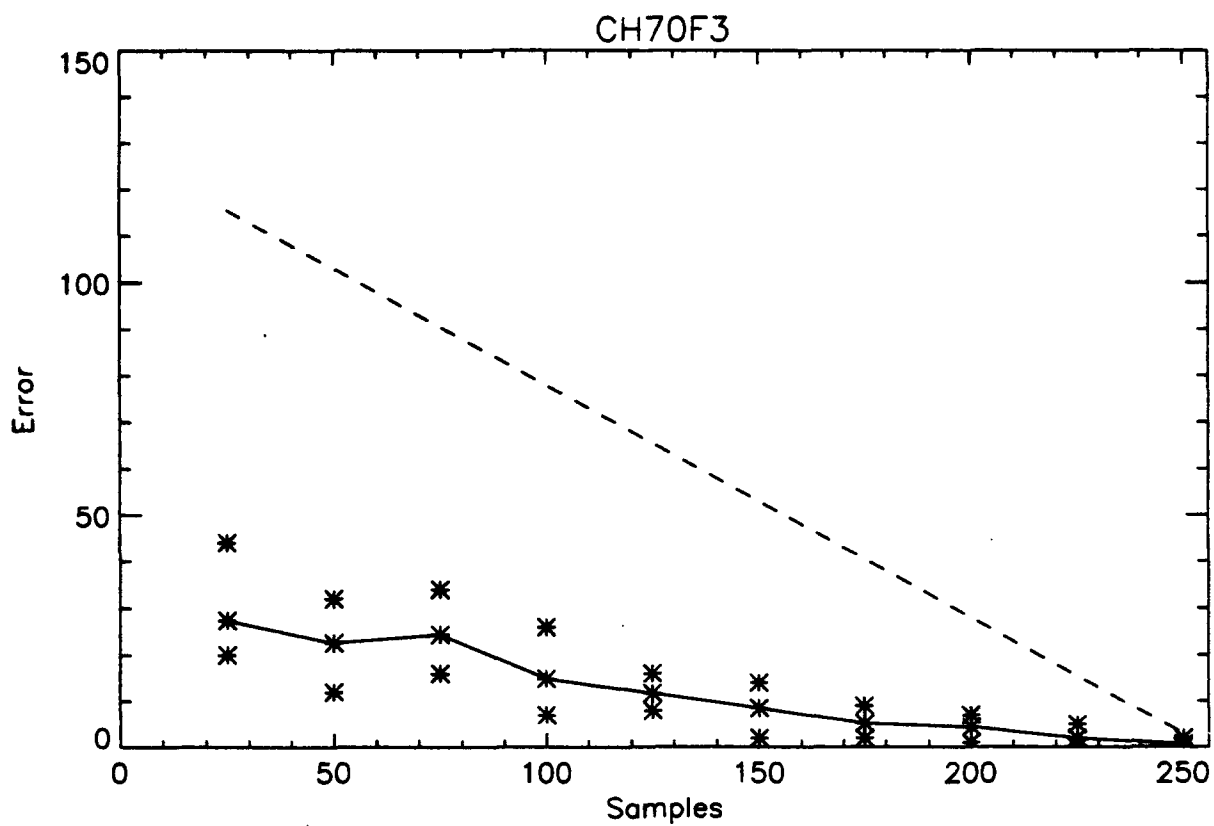
—*— Avg Error

CH52F4



- Chance
- * Max error
- * Min error
- *--- Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



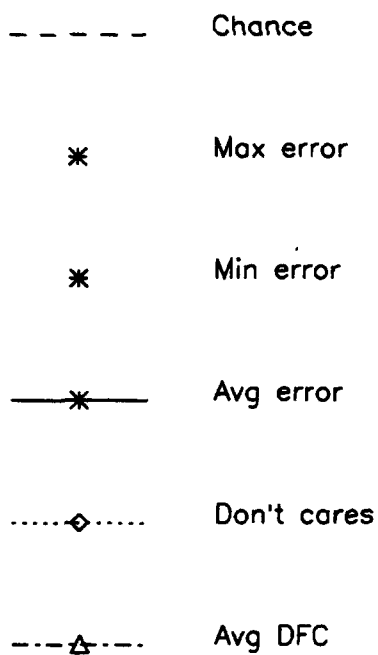
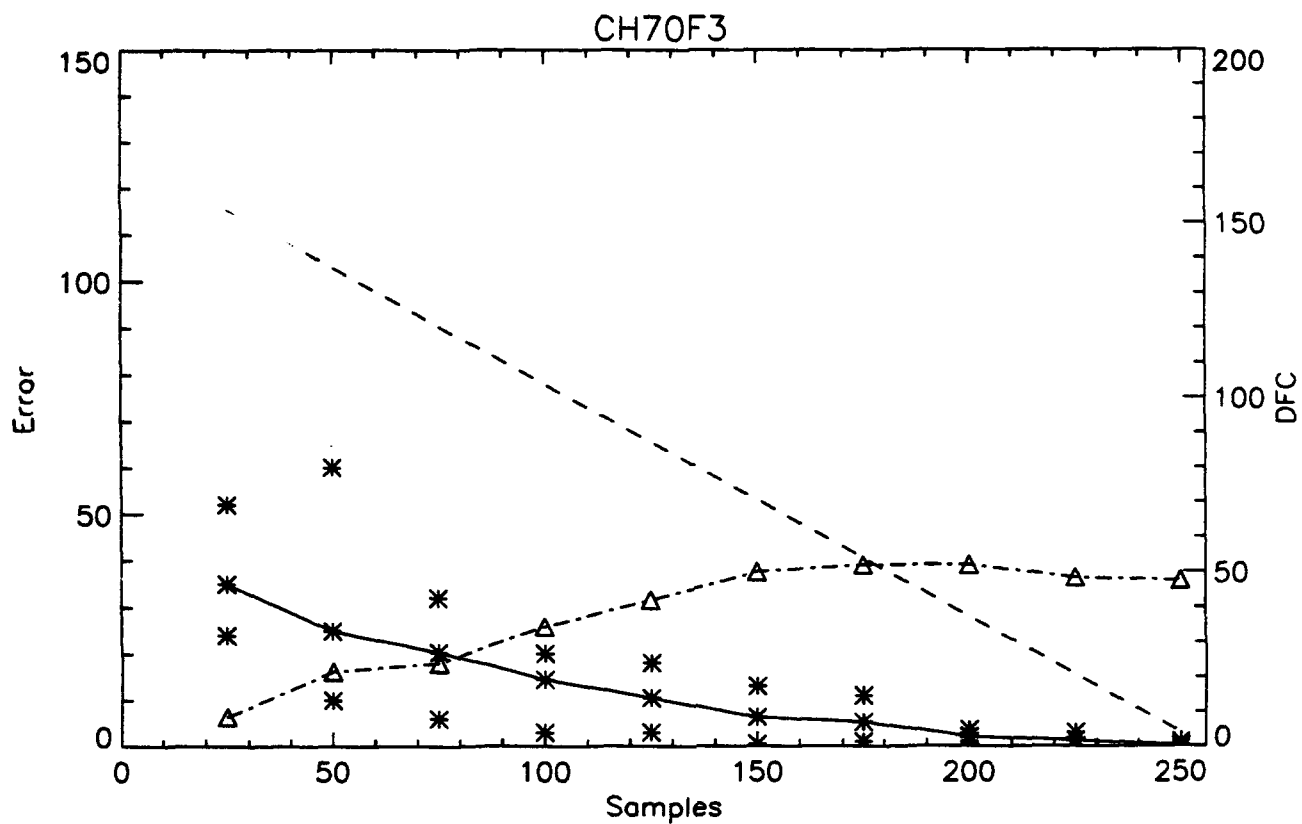
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

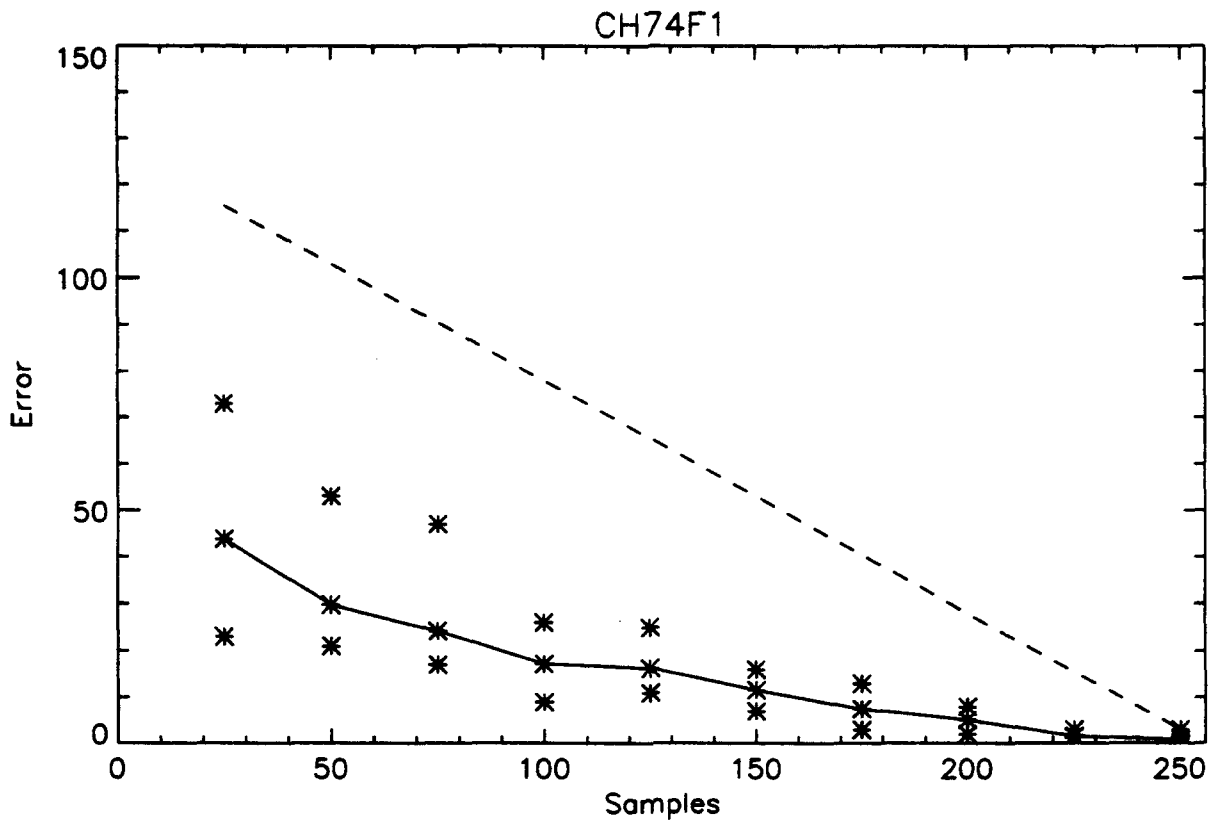
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



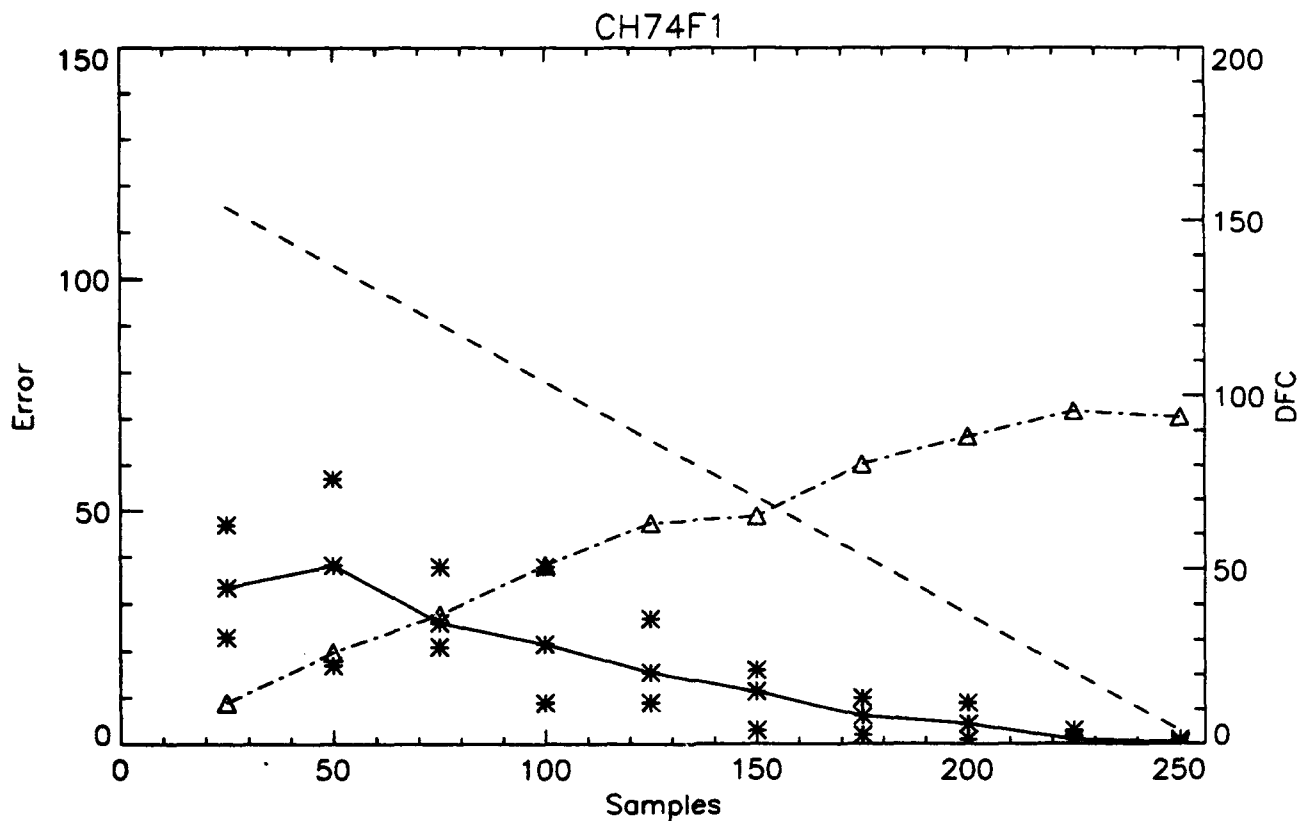
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

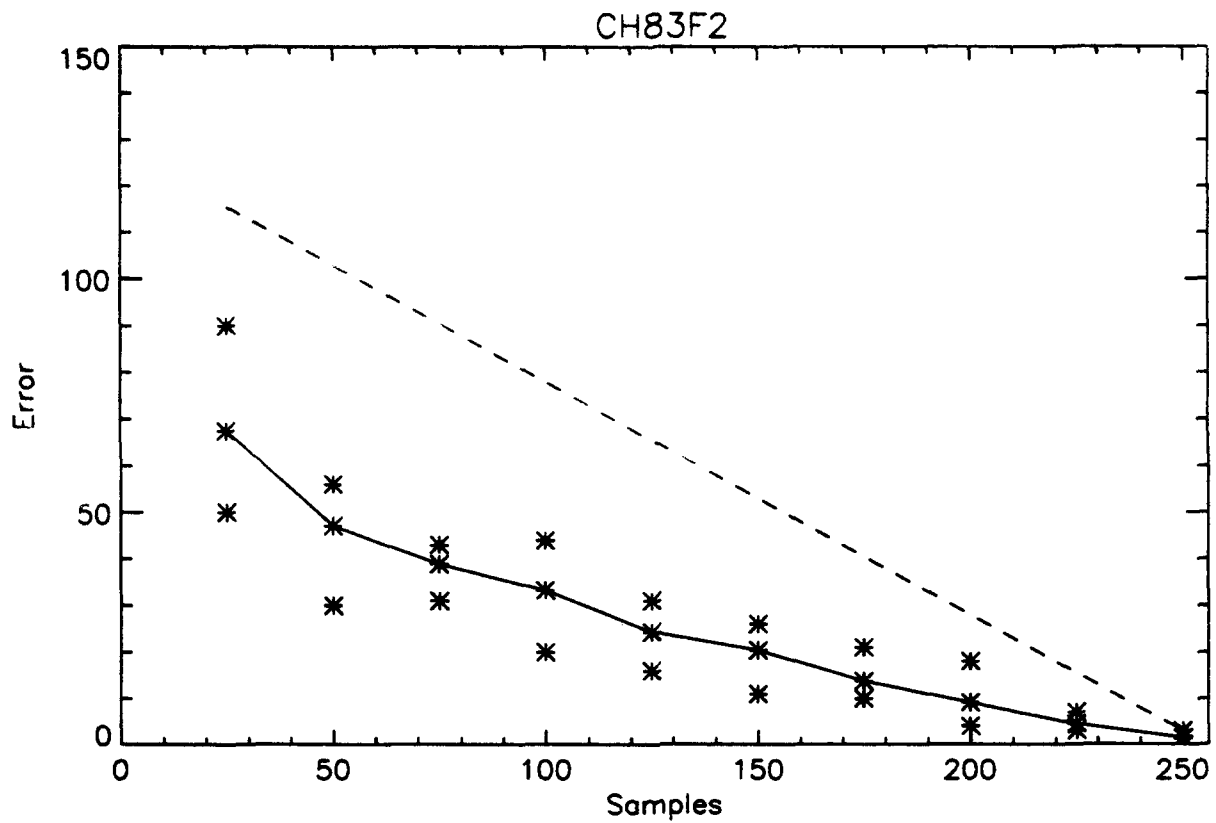
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



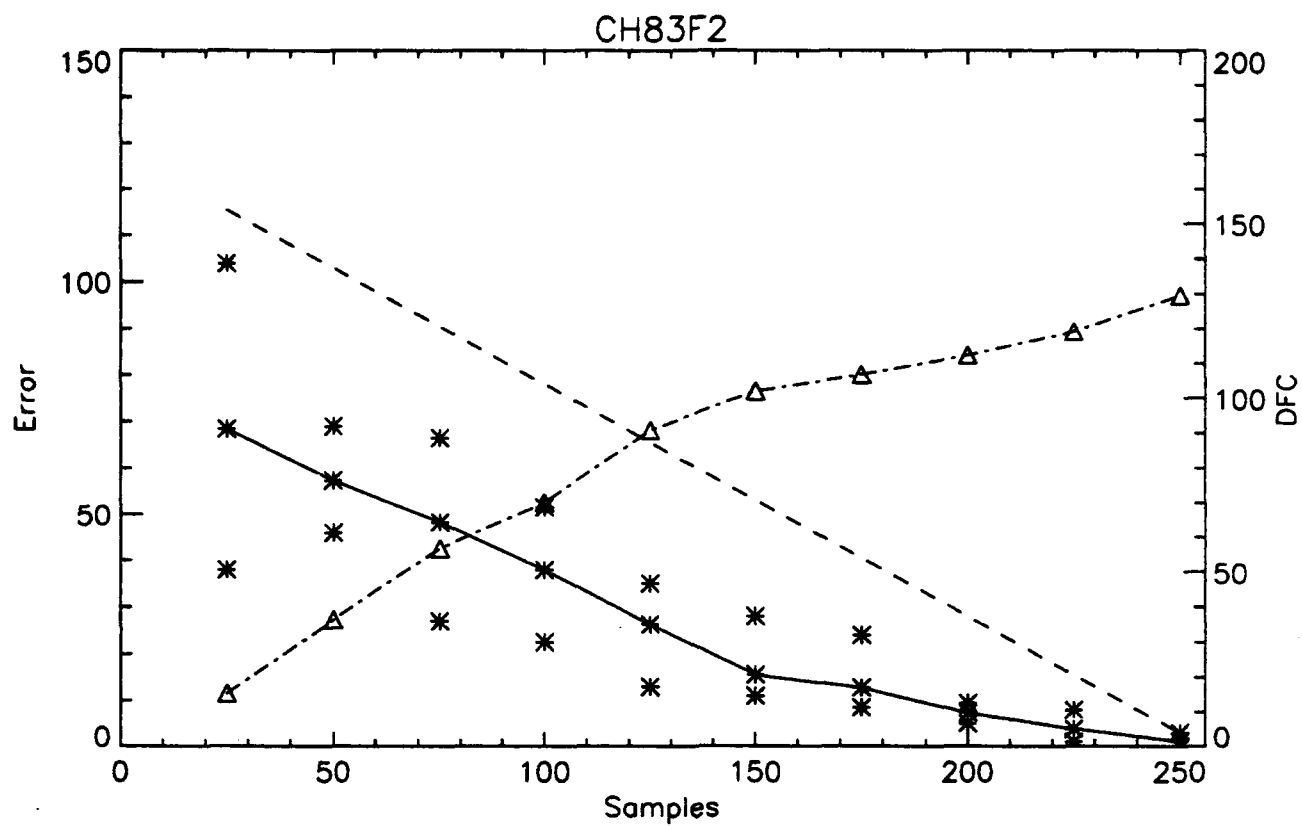
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

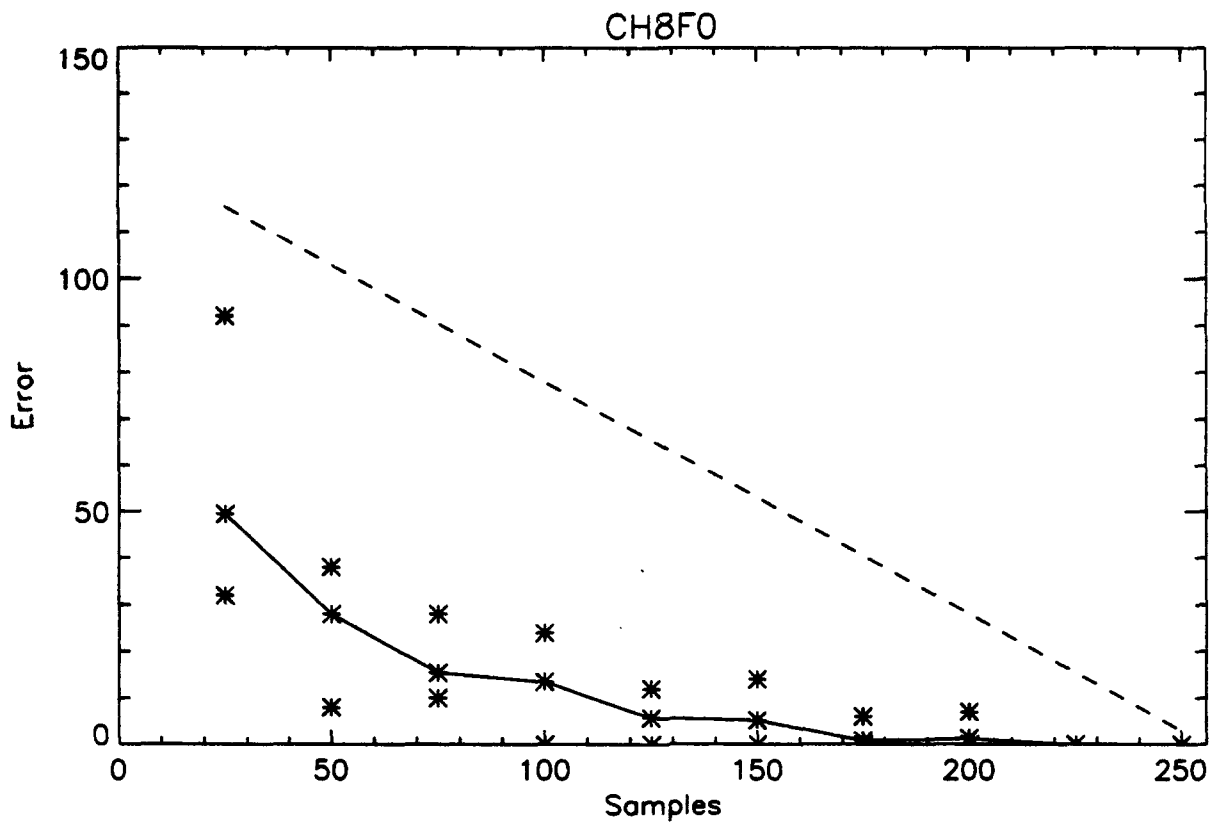
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



--- Chance

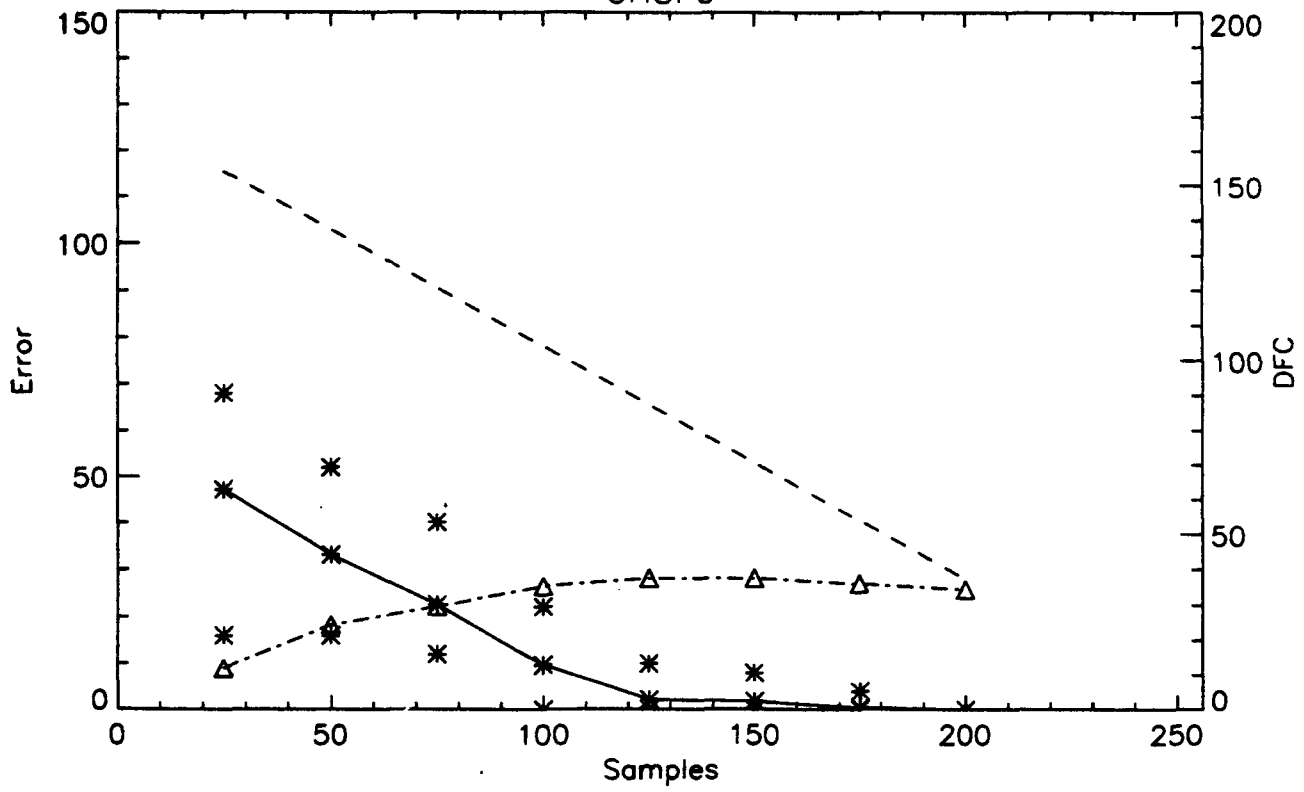
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

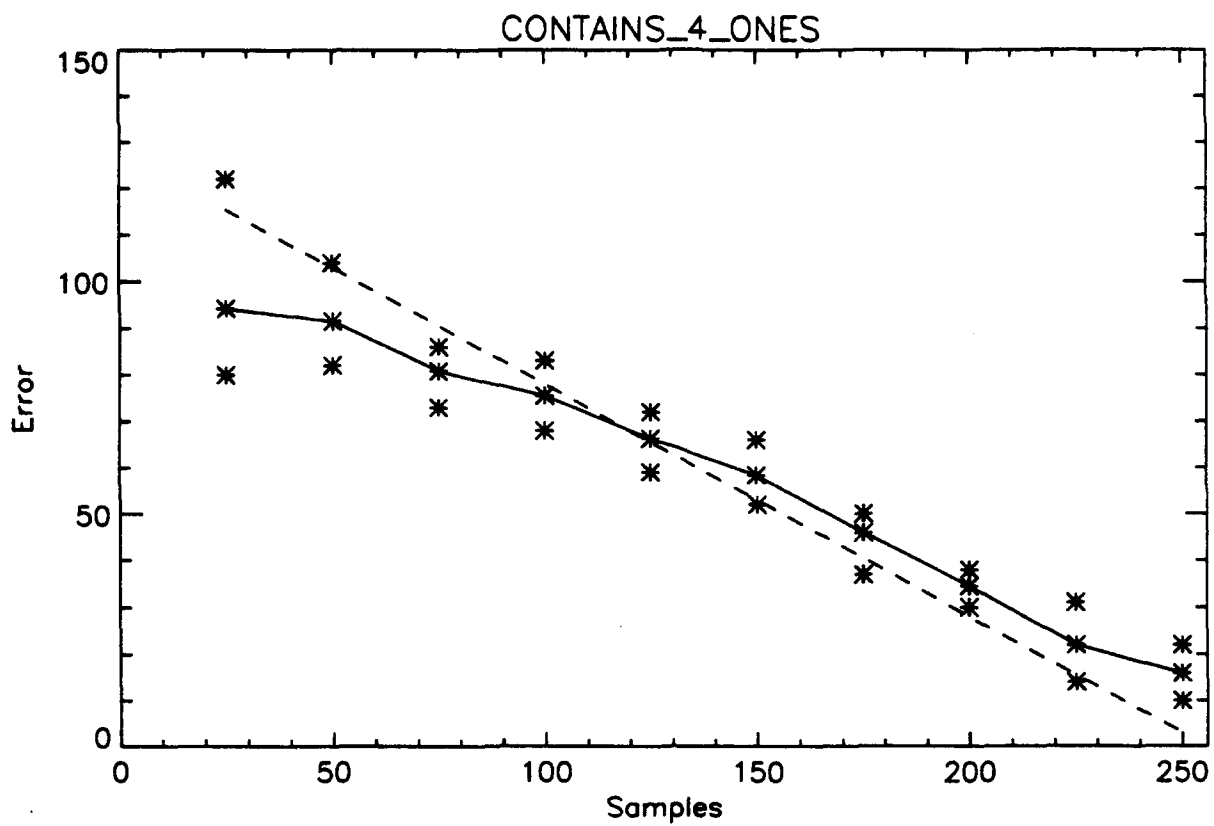
—* Avg Error

CH8F0



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



----- Chance

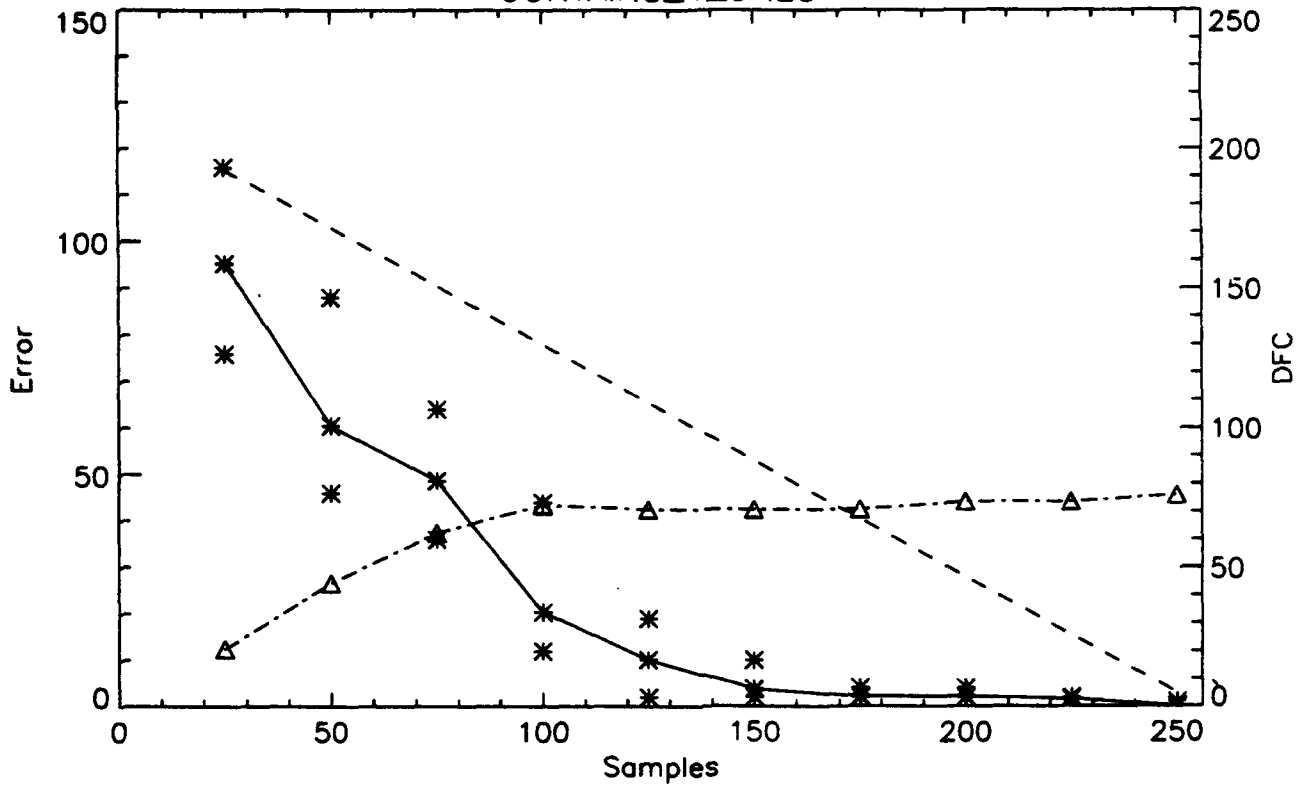
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

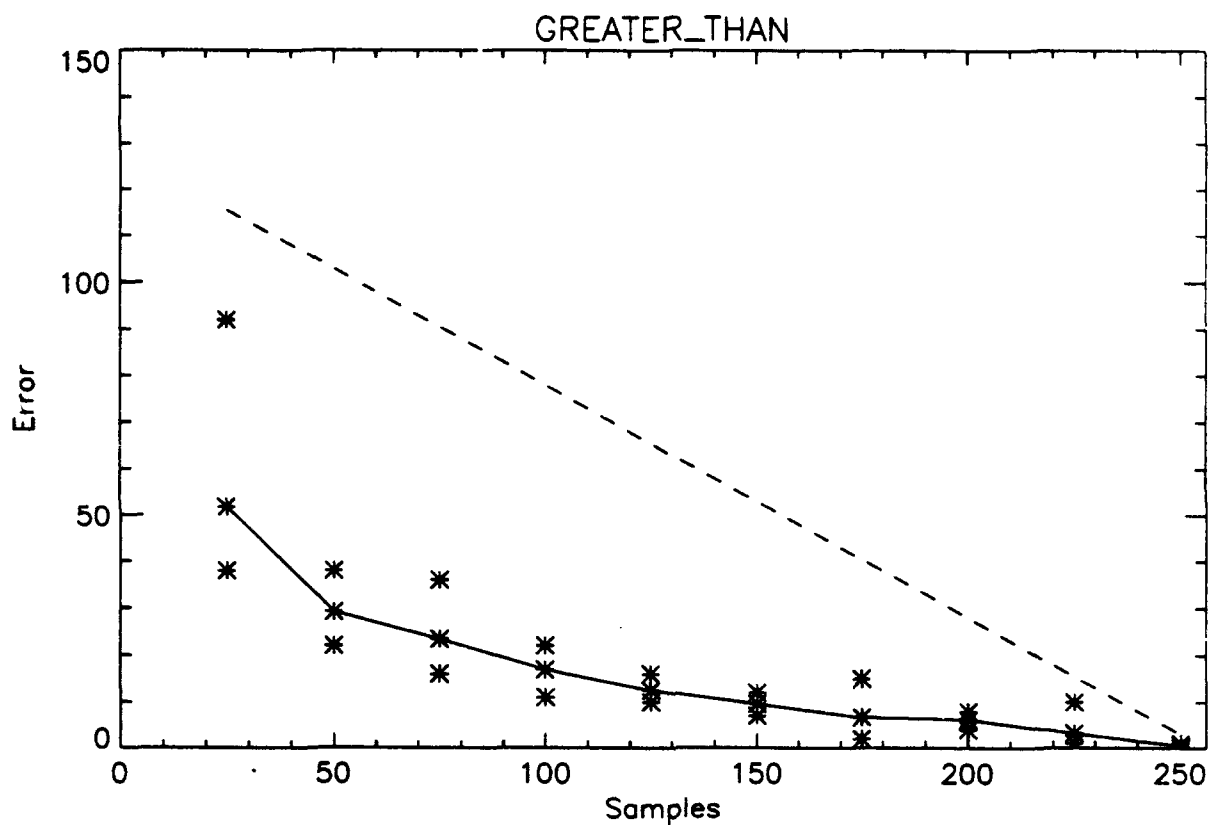
—*— Avg Error

CONTAINS_4_ONES



- Chance
- * Max error
- * Min error
- *--- Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



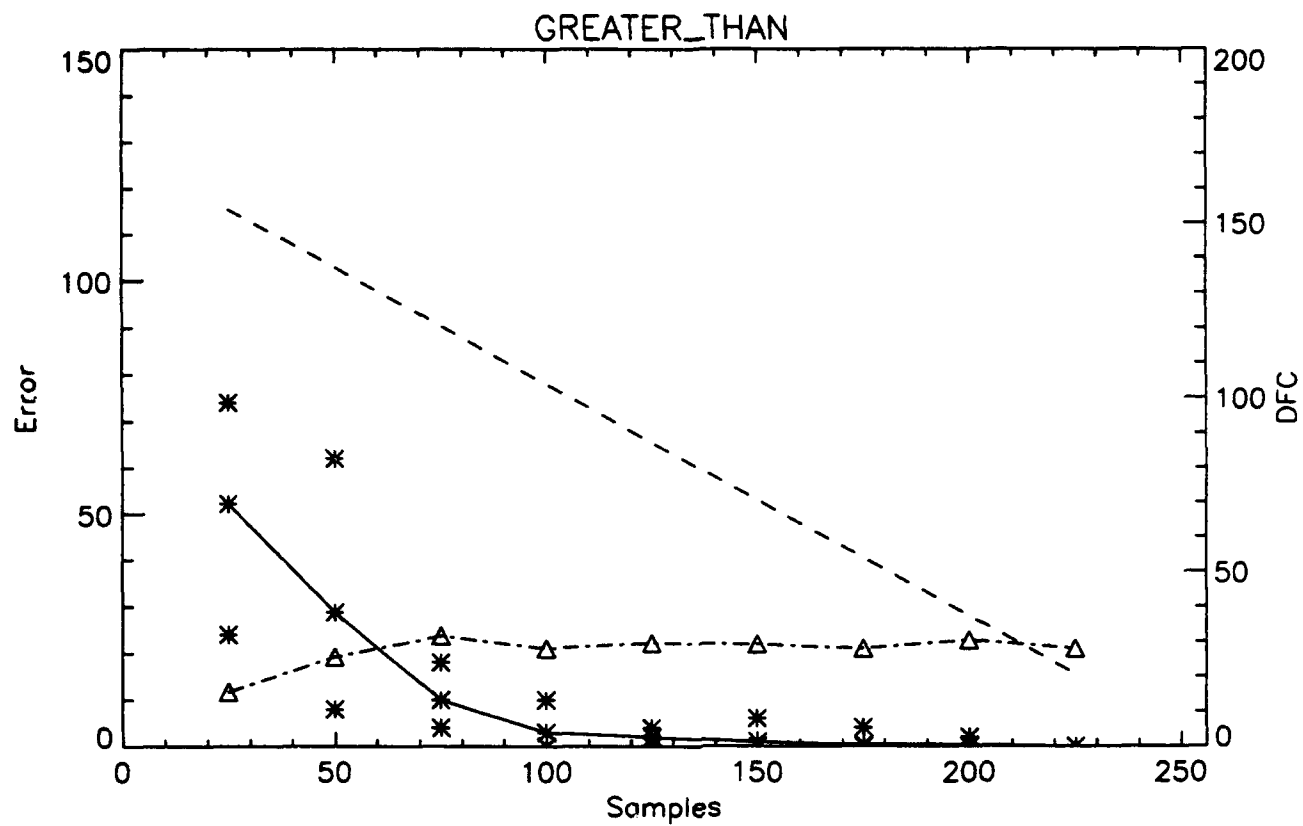
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

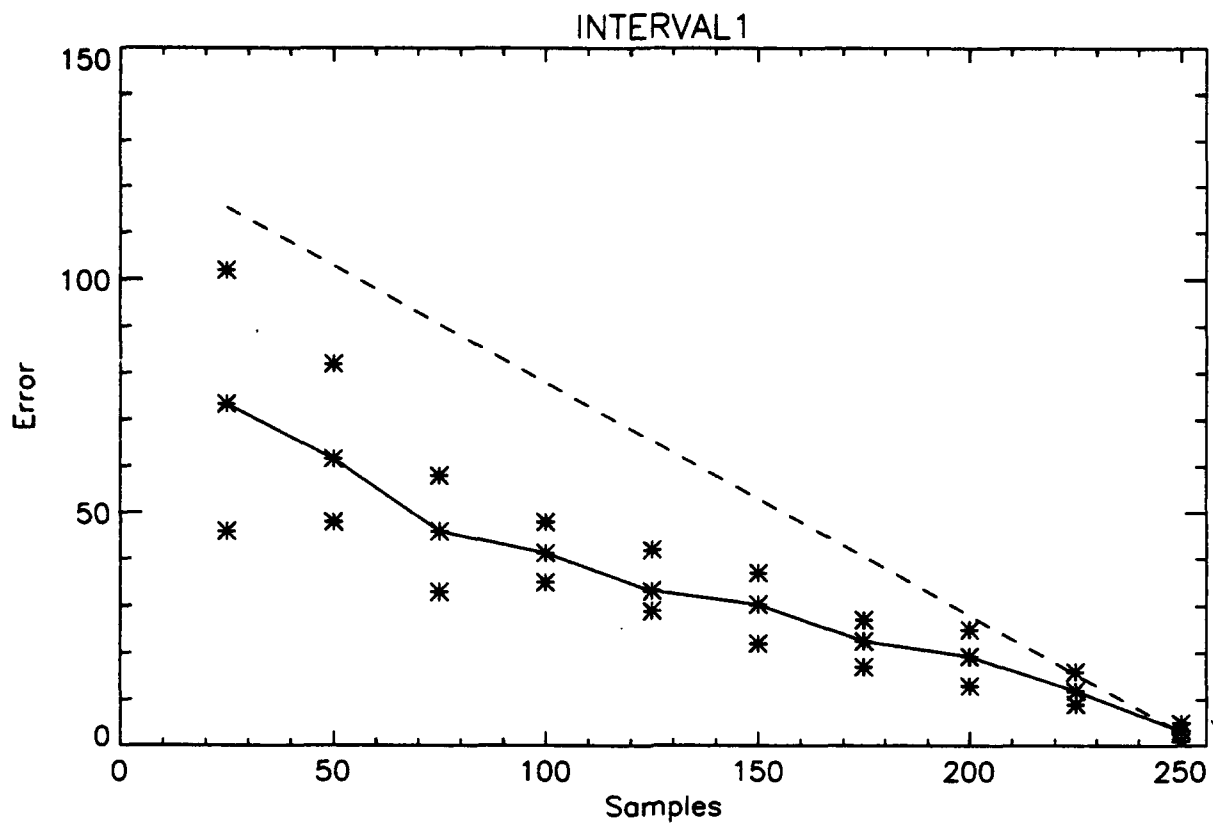
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-△-.- Avg D C

FLASH dni0e300



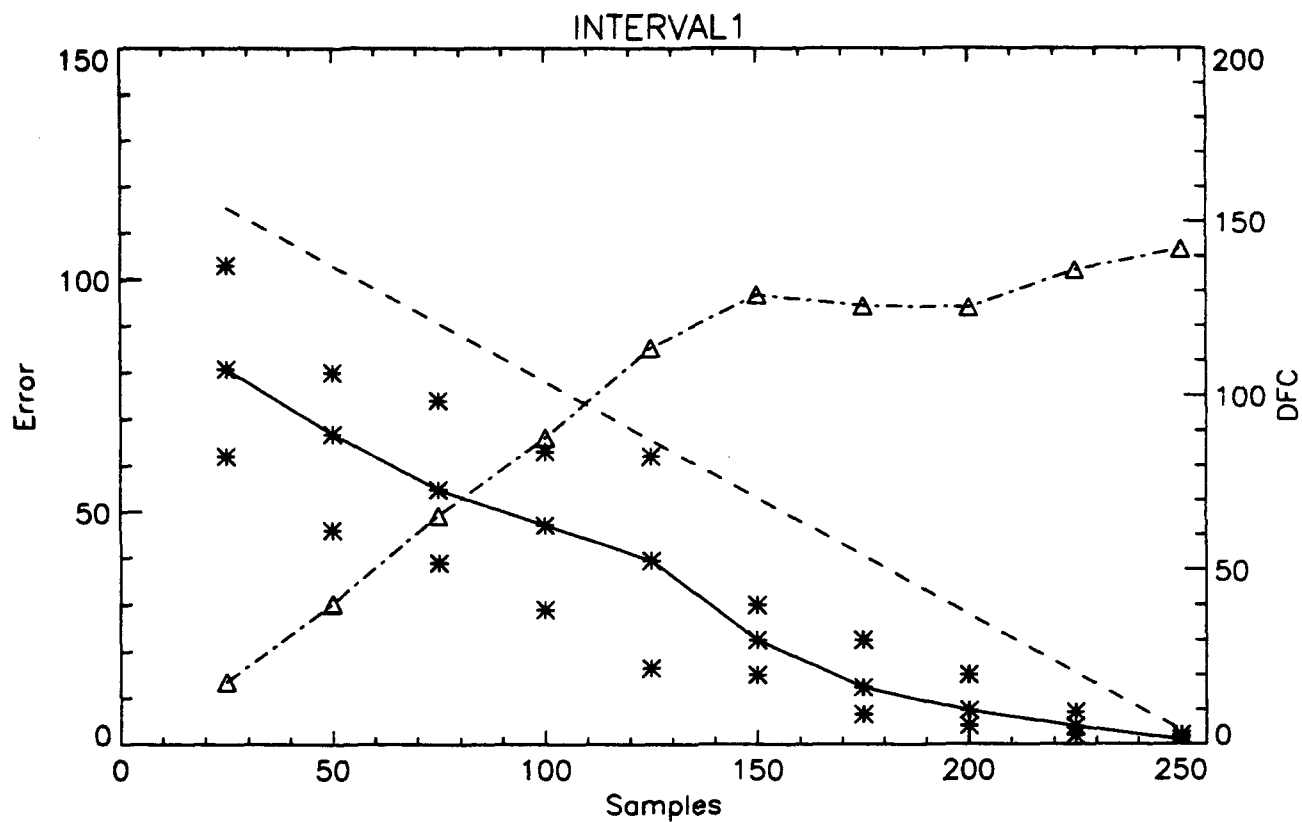
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

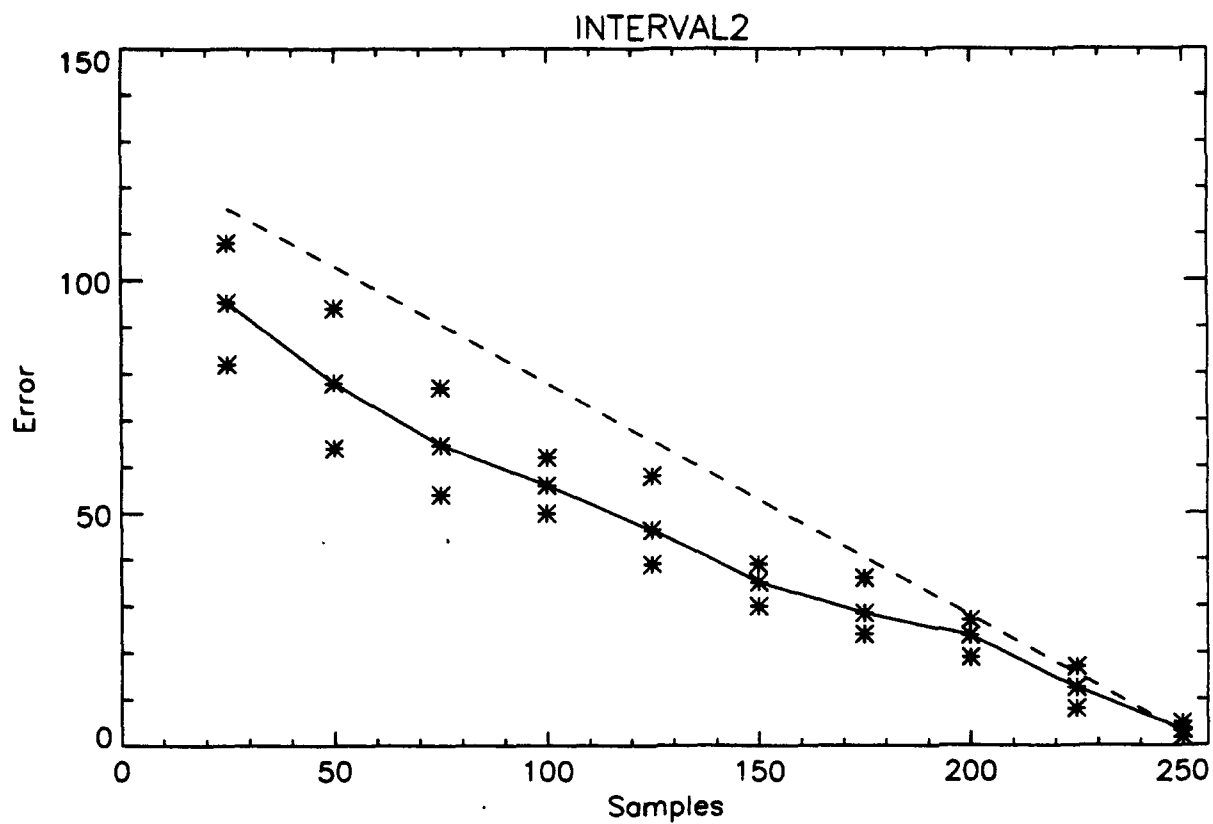
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-.-△-.-.- Avg DFC

FLASH dni0e300



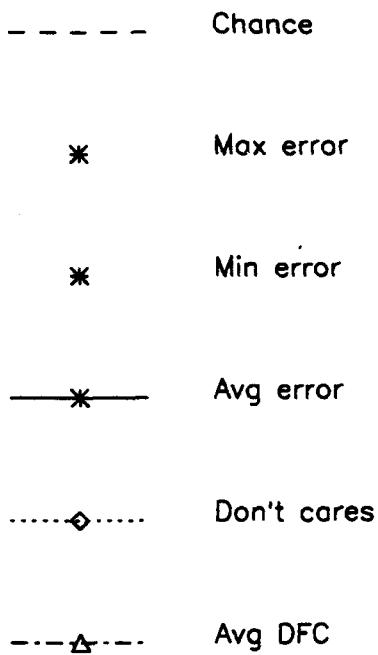
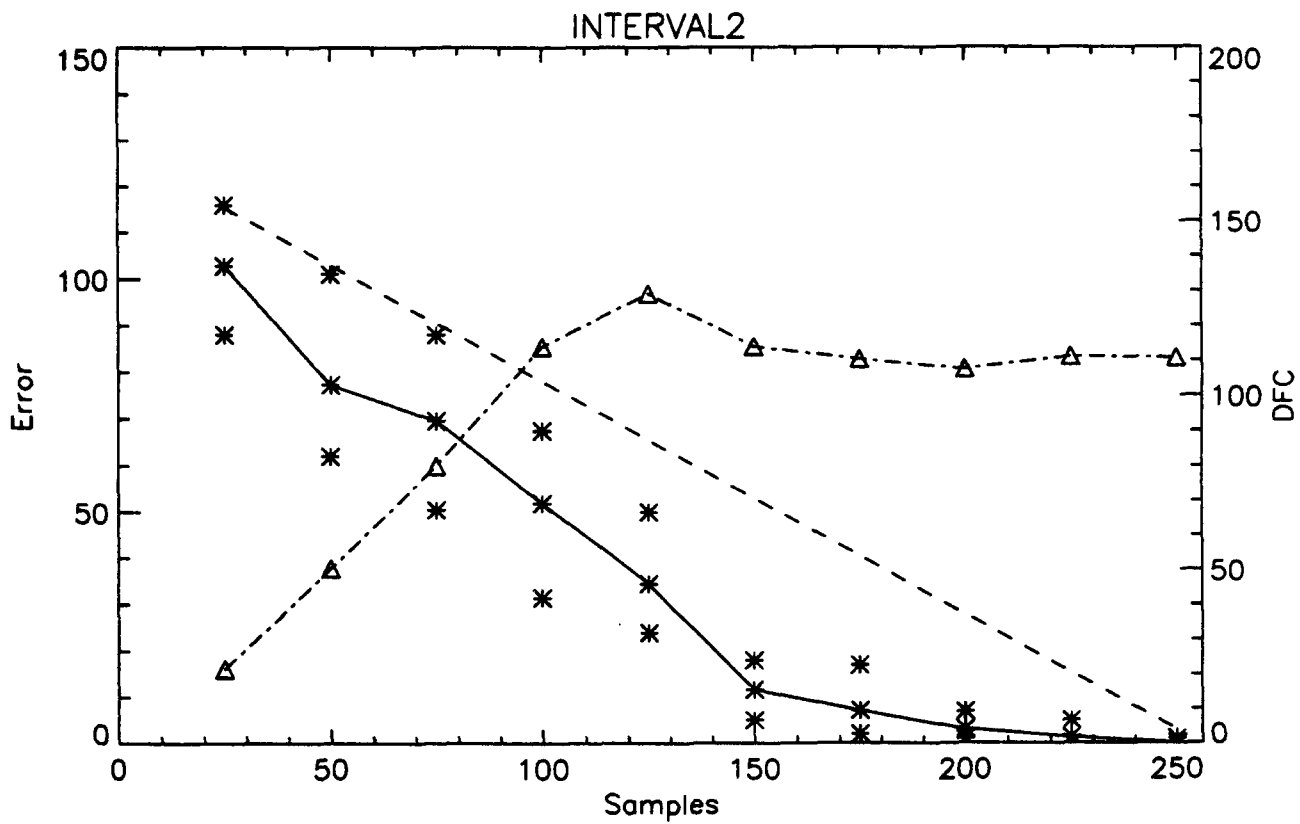
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

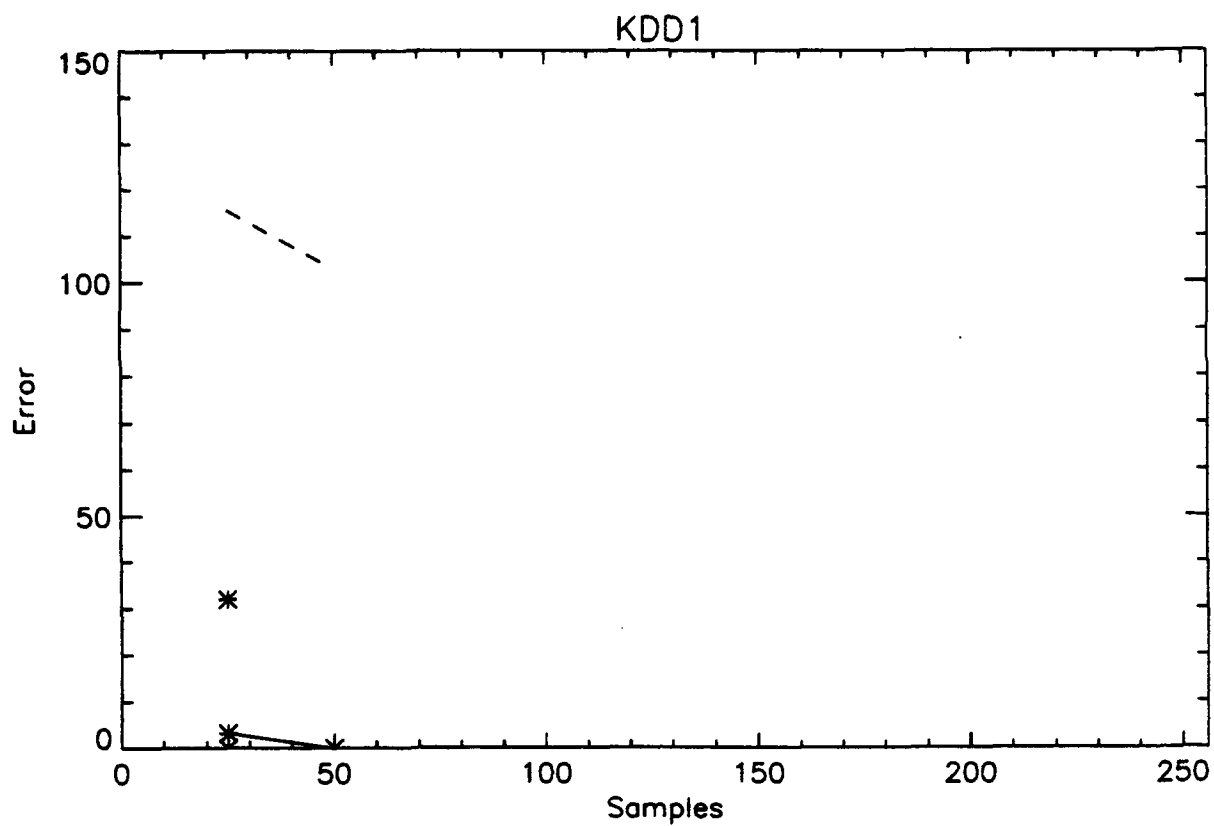
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



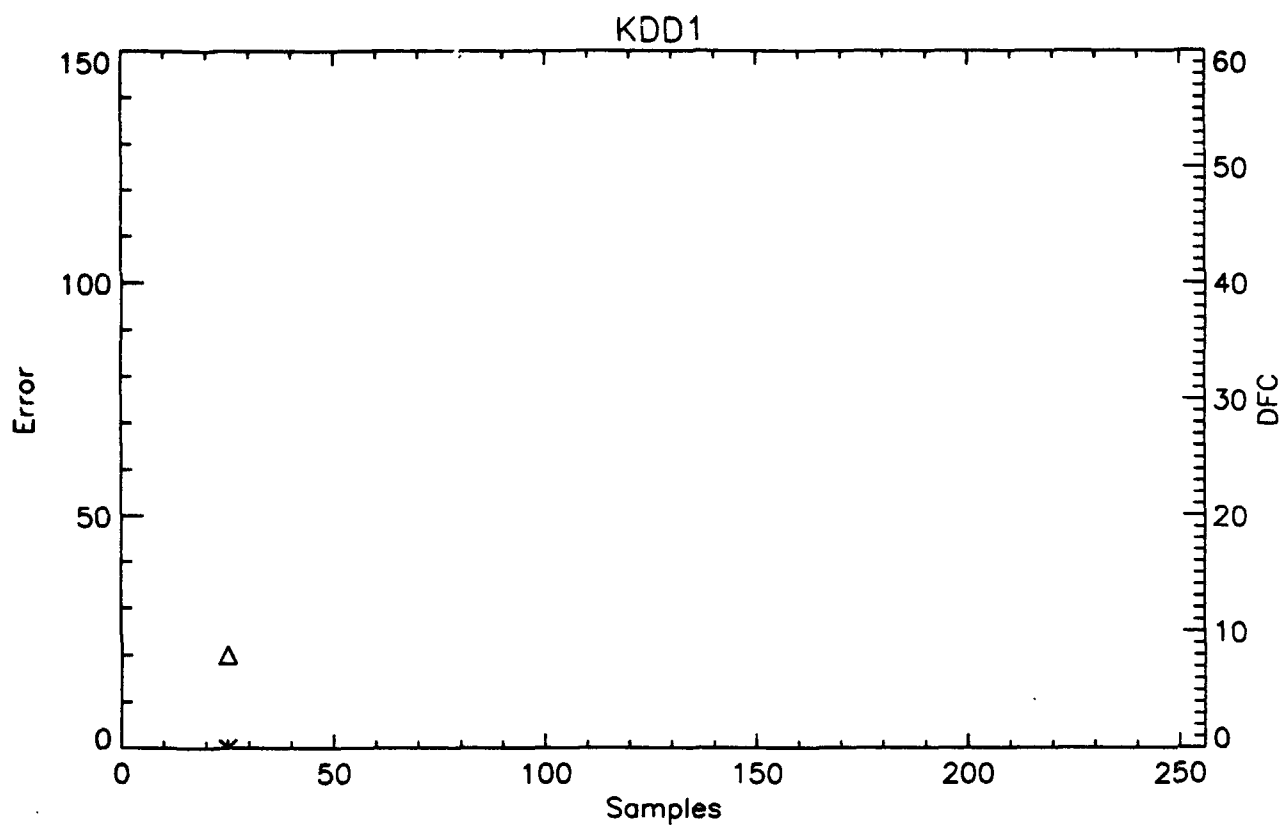
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

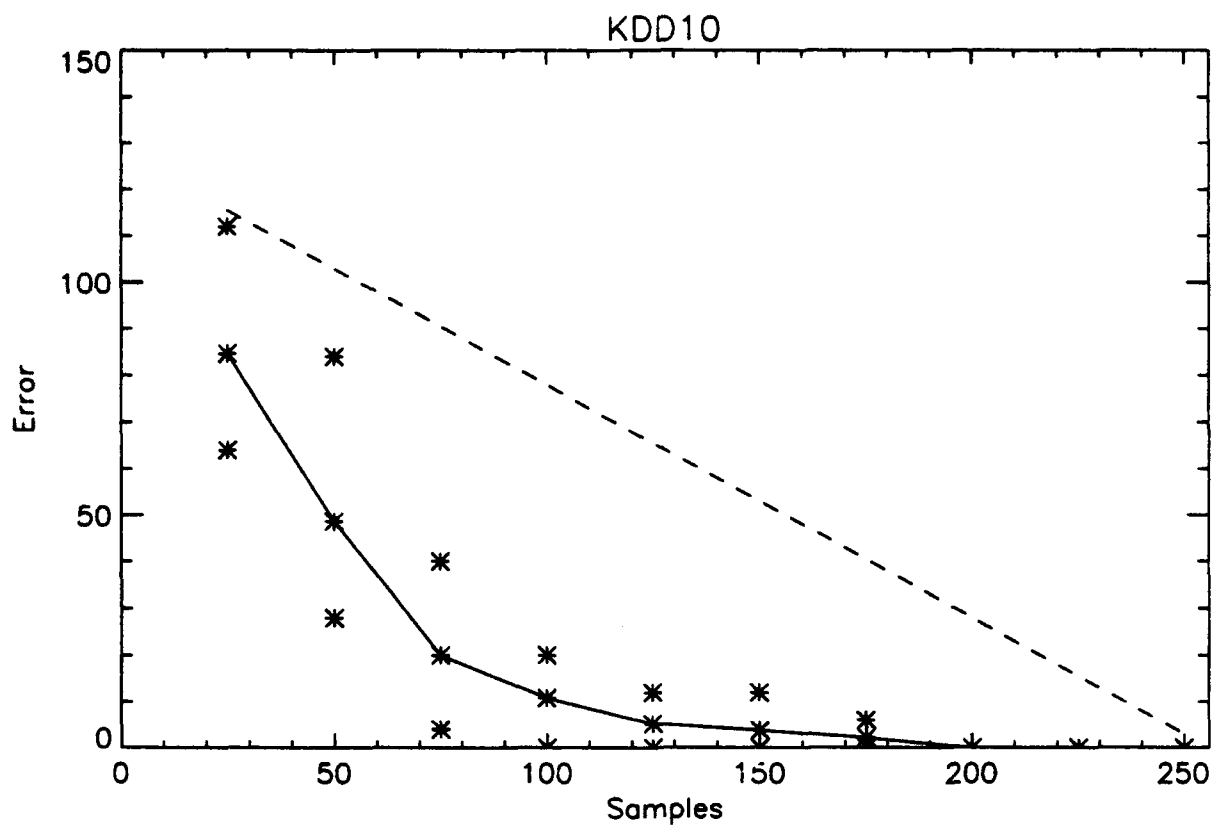
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
- ...◇... Don't cares
- △--- Avg DFC

FLASH dni0e300



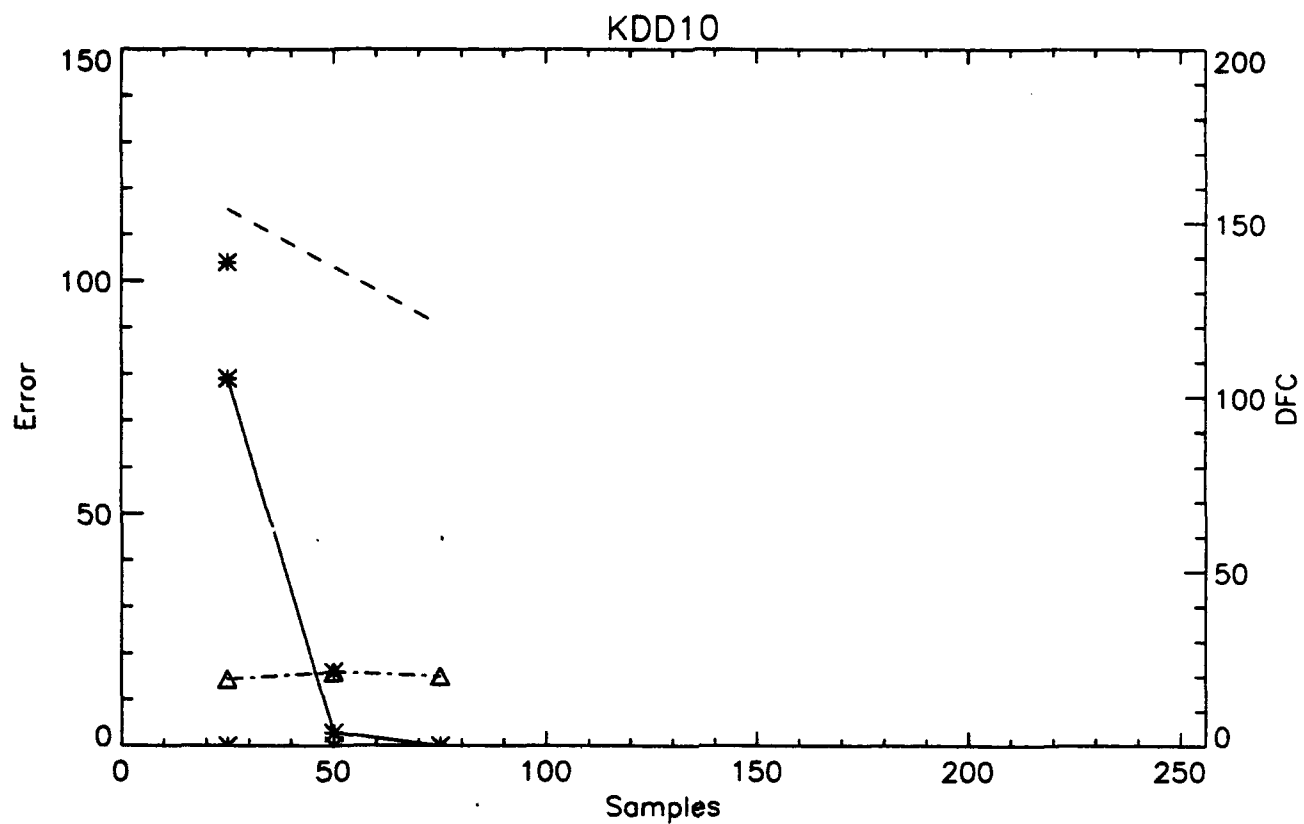
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

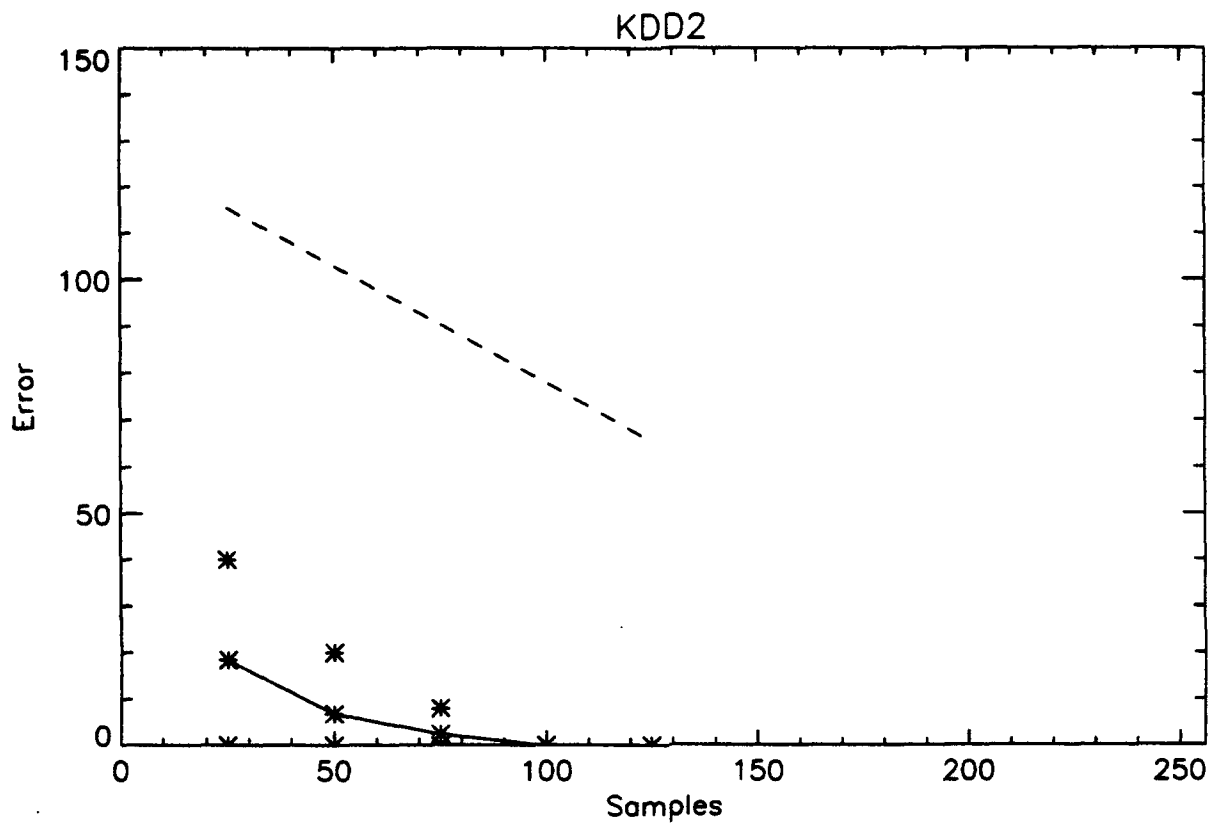
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



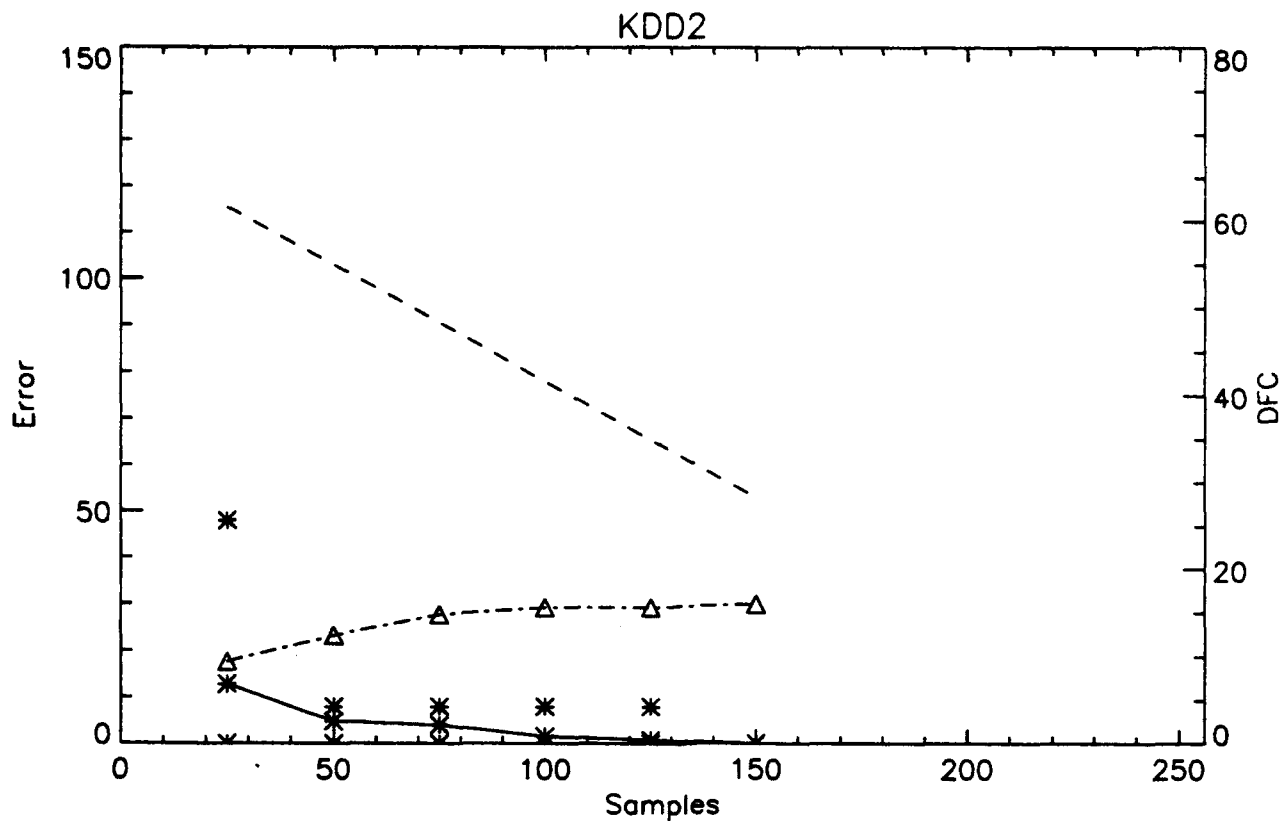
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

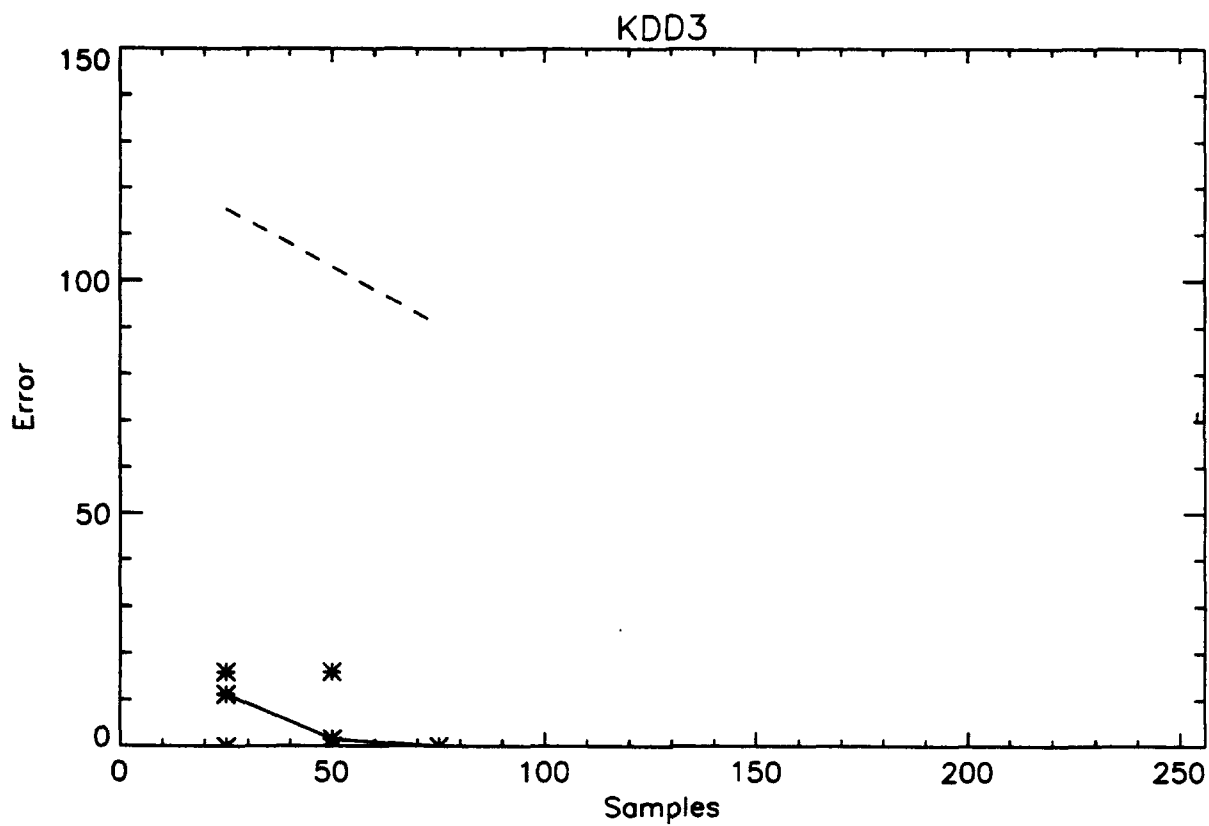
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



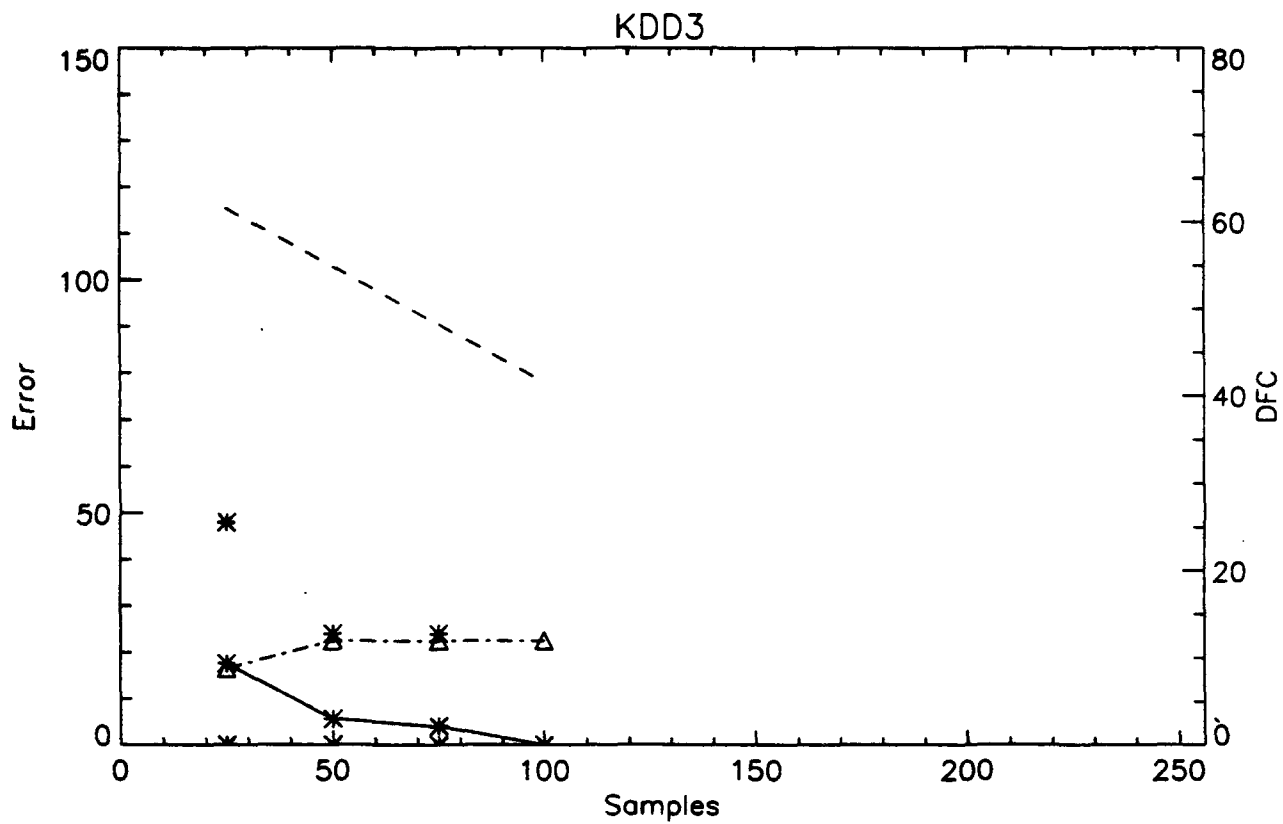
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

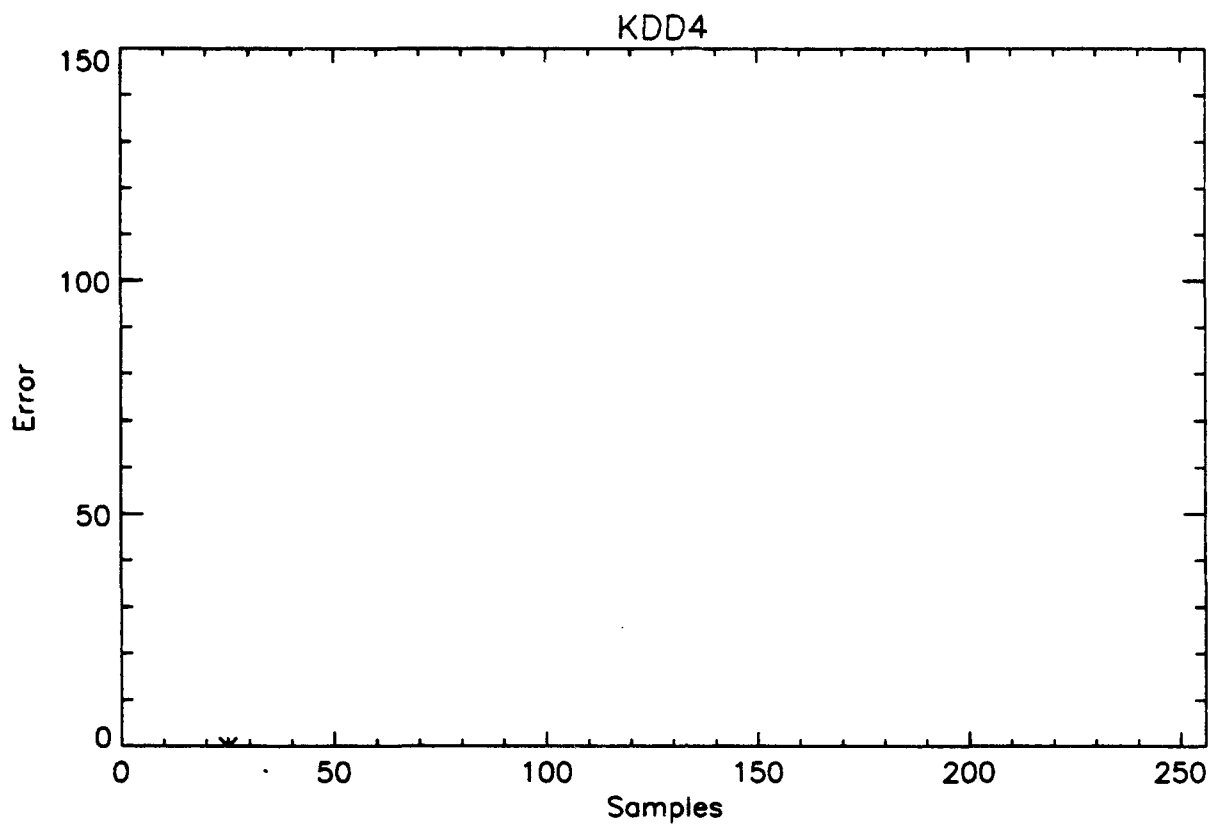
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



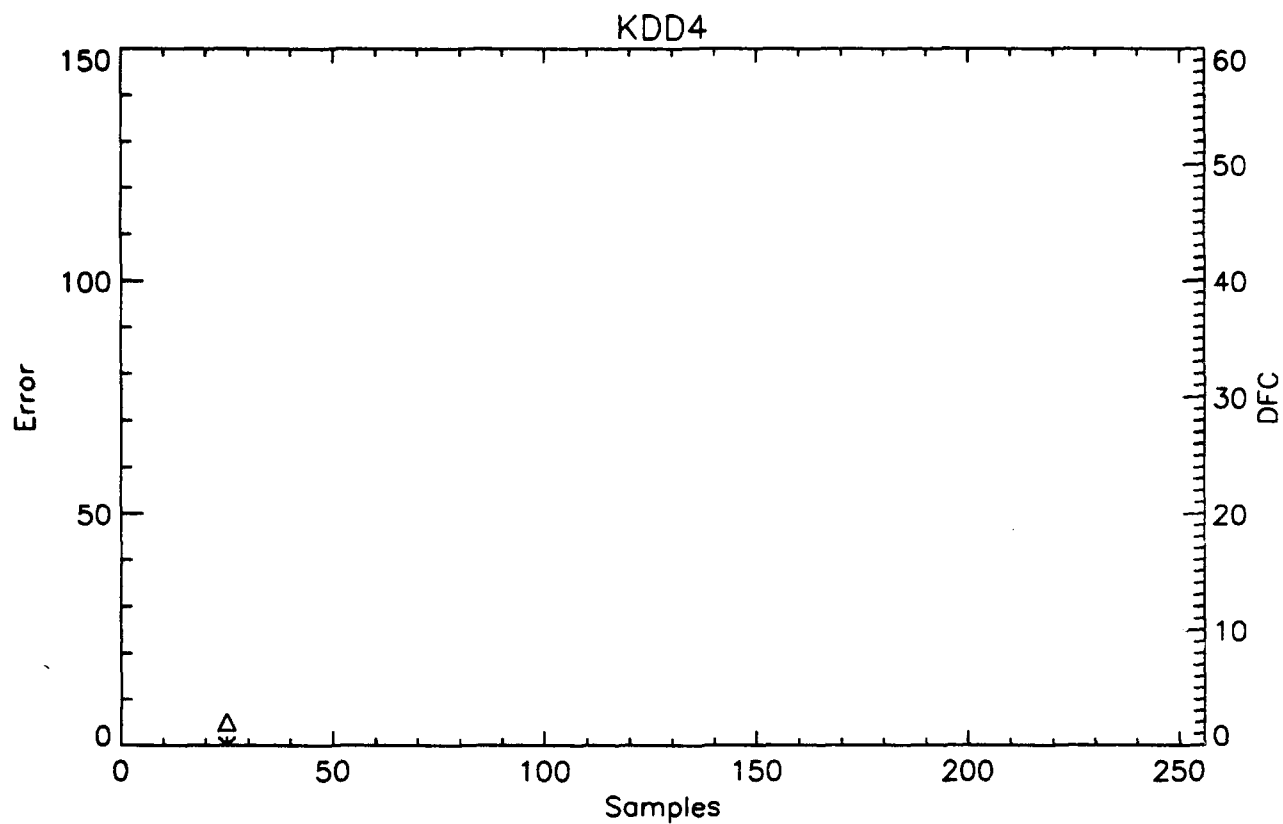
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

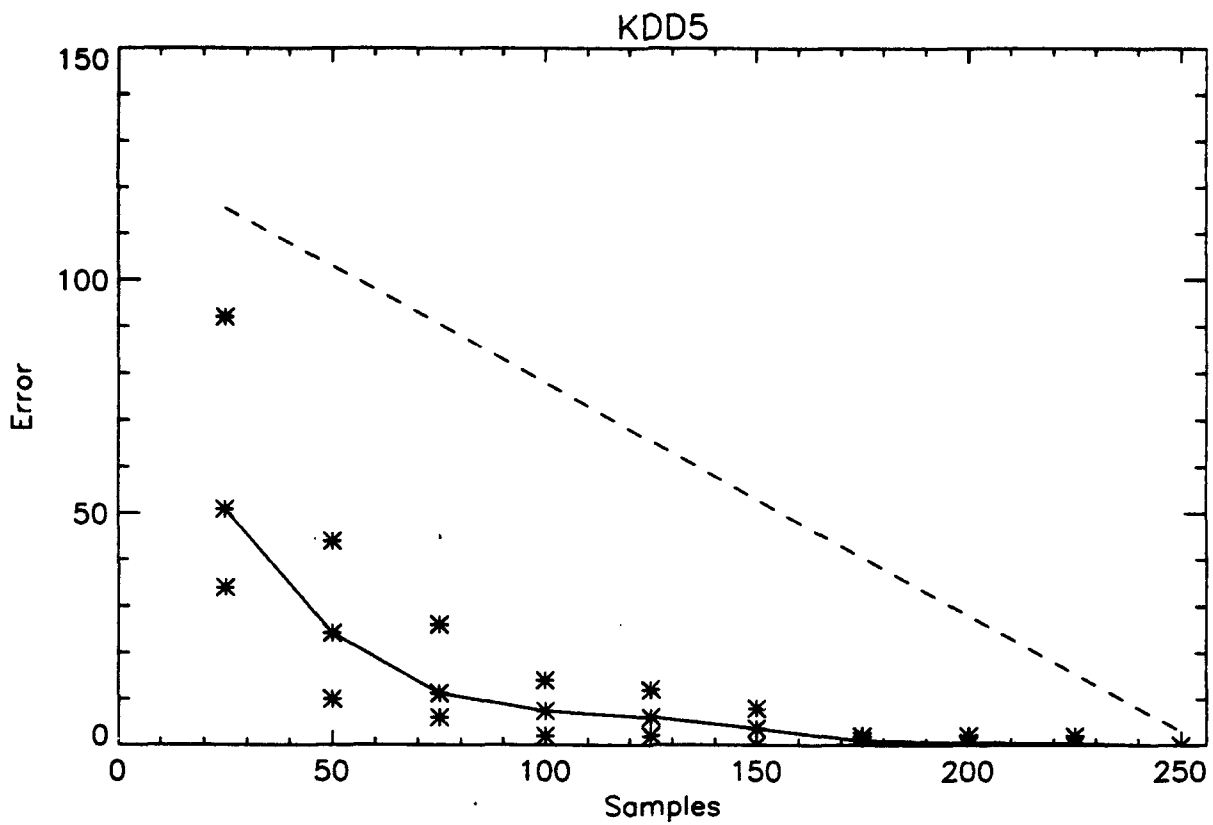
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



----- Chance

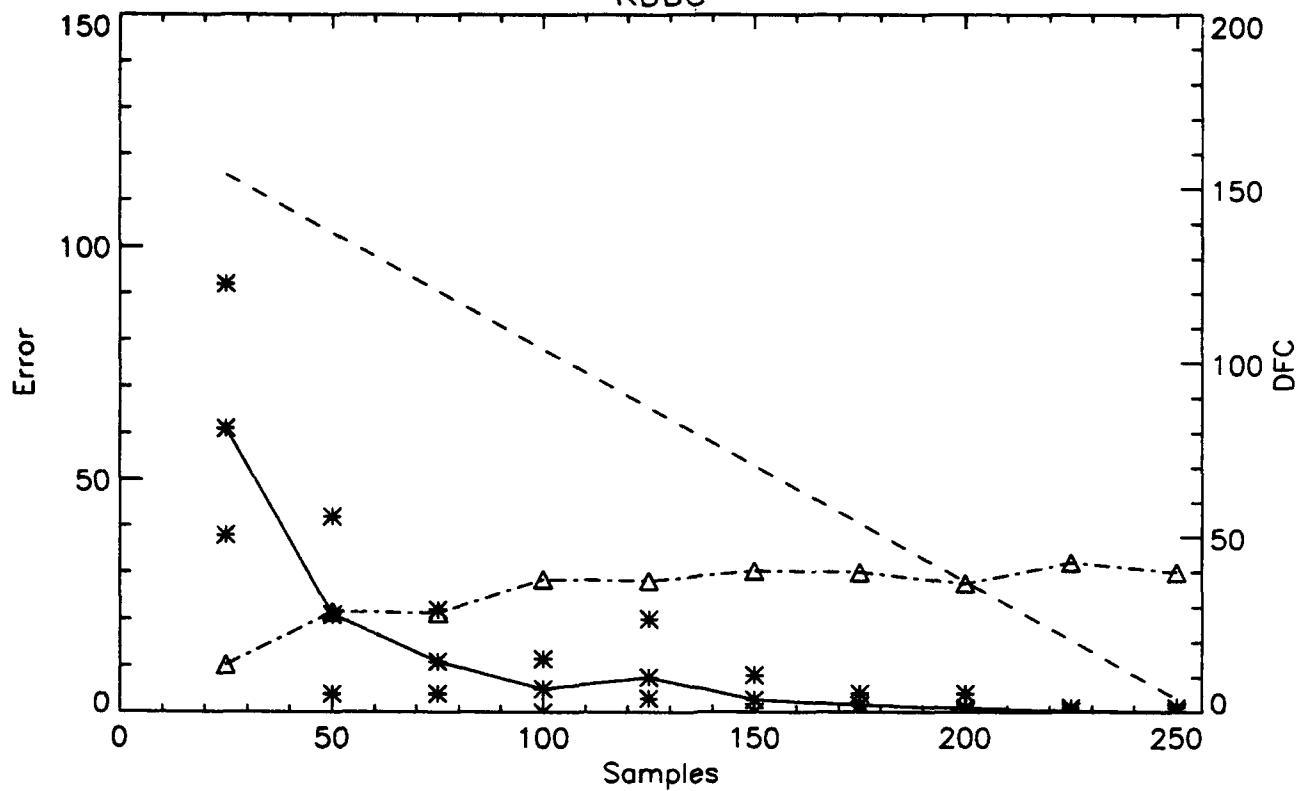
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

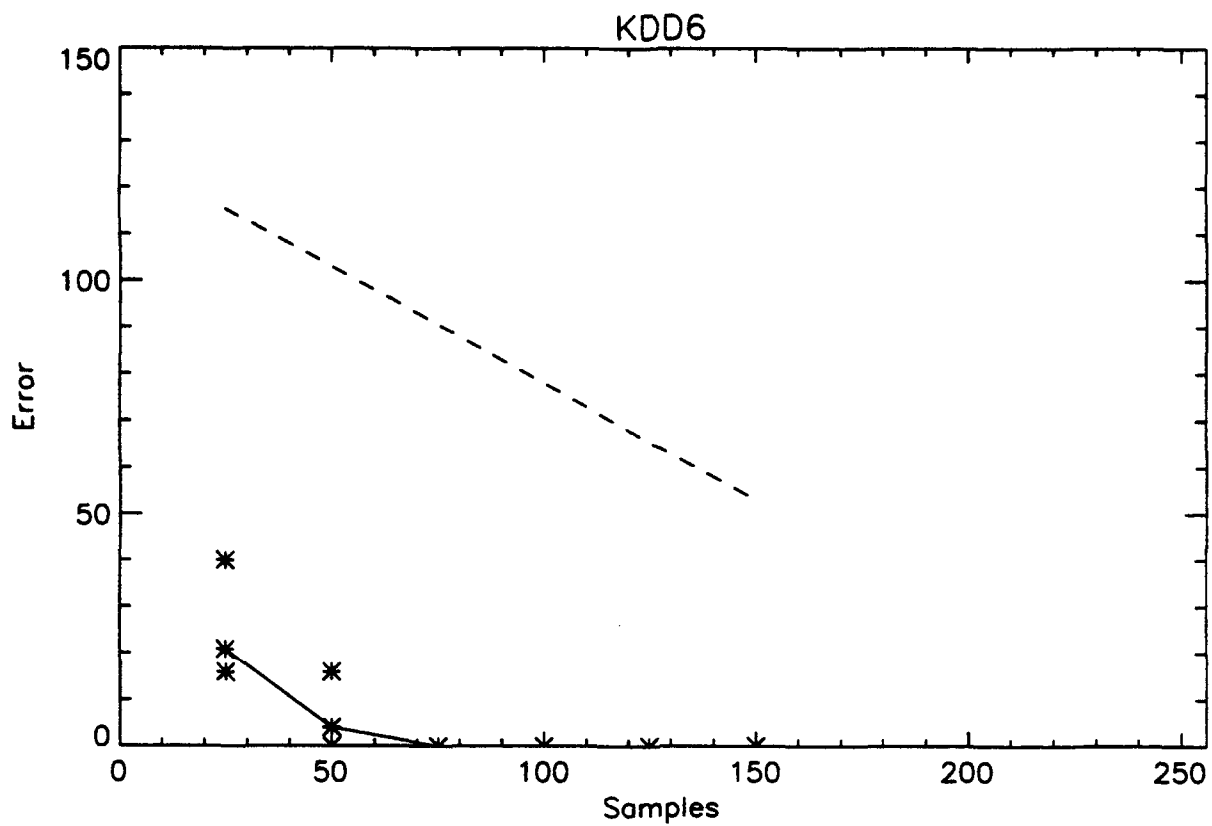
—*— Avg Error

KDD5



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



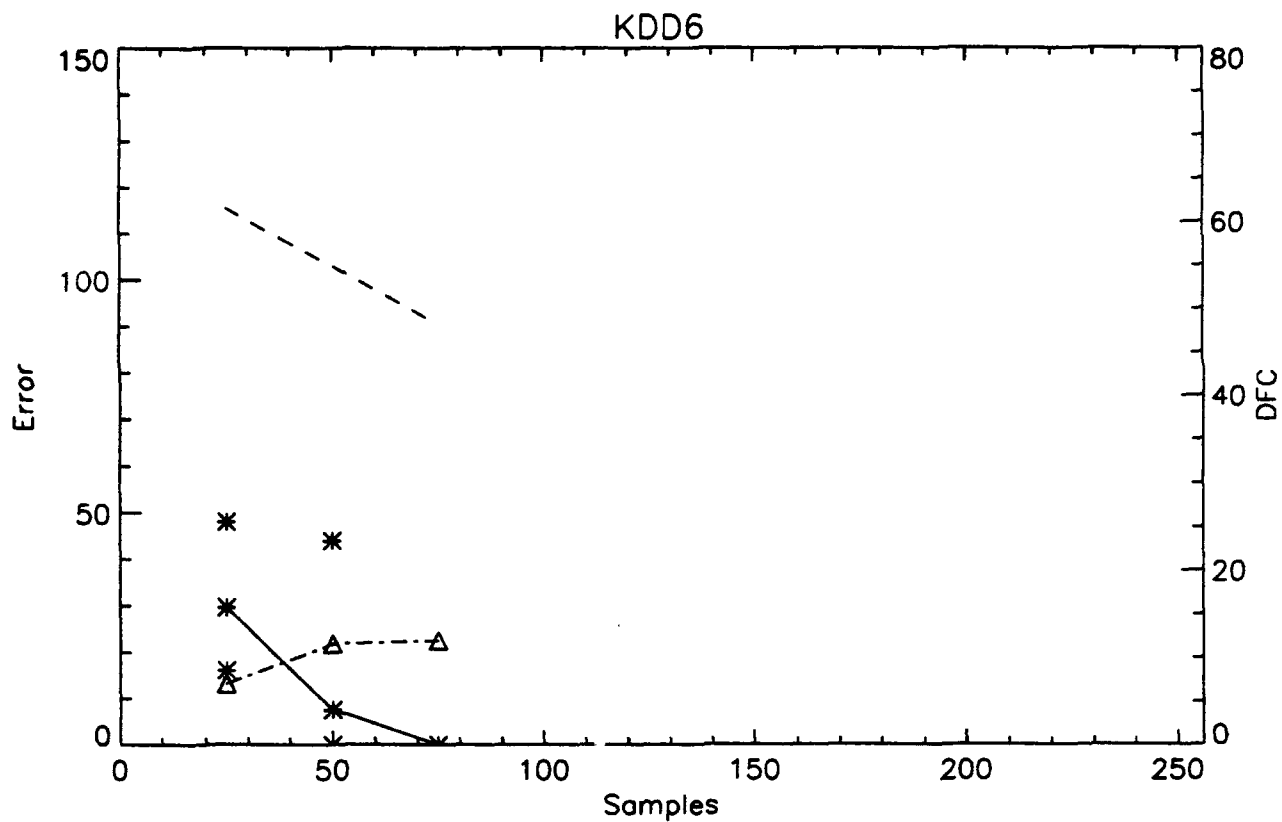
--- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

—*— Avg Error



- Chance

- * Max error

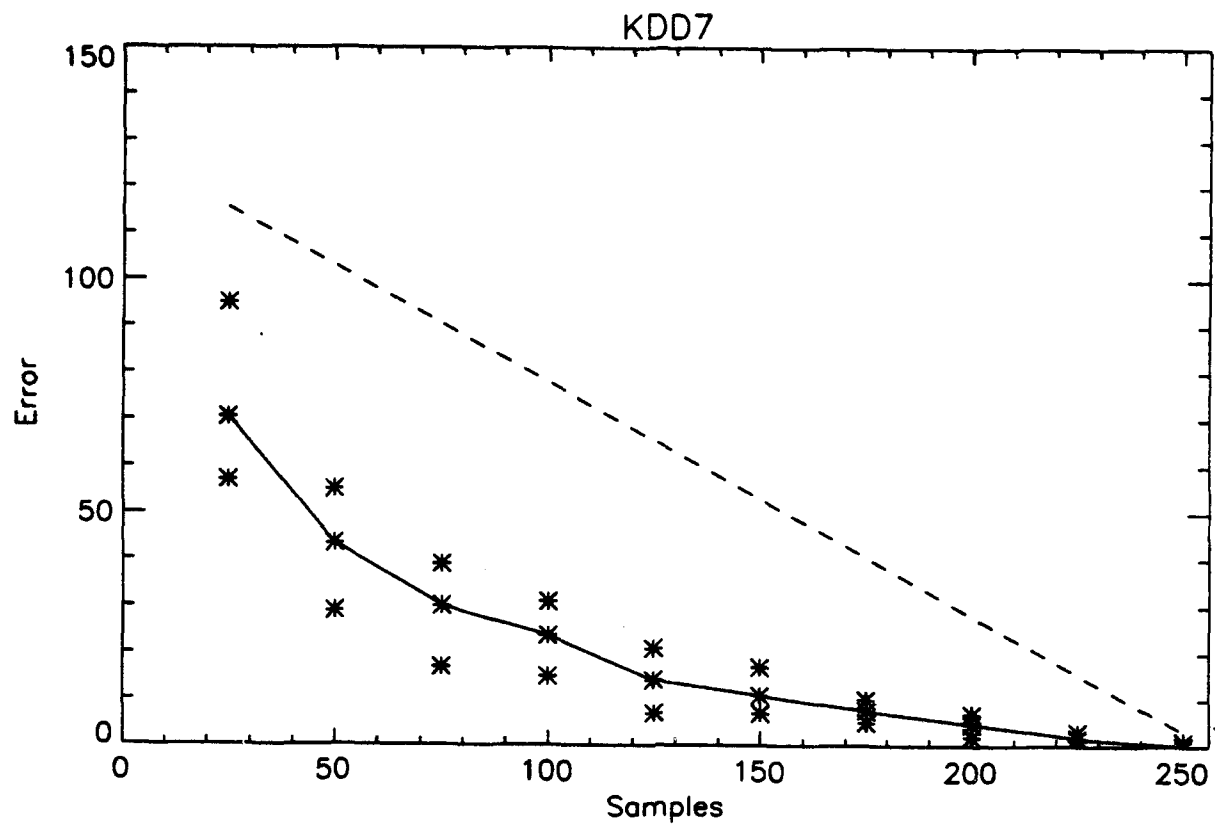
- * Min error

- *— Avg error

-◇..... Don't cares

- △--- Avg DFC

FLASH dni0e300



----- Chance

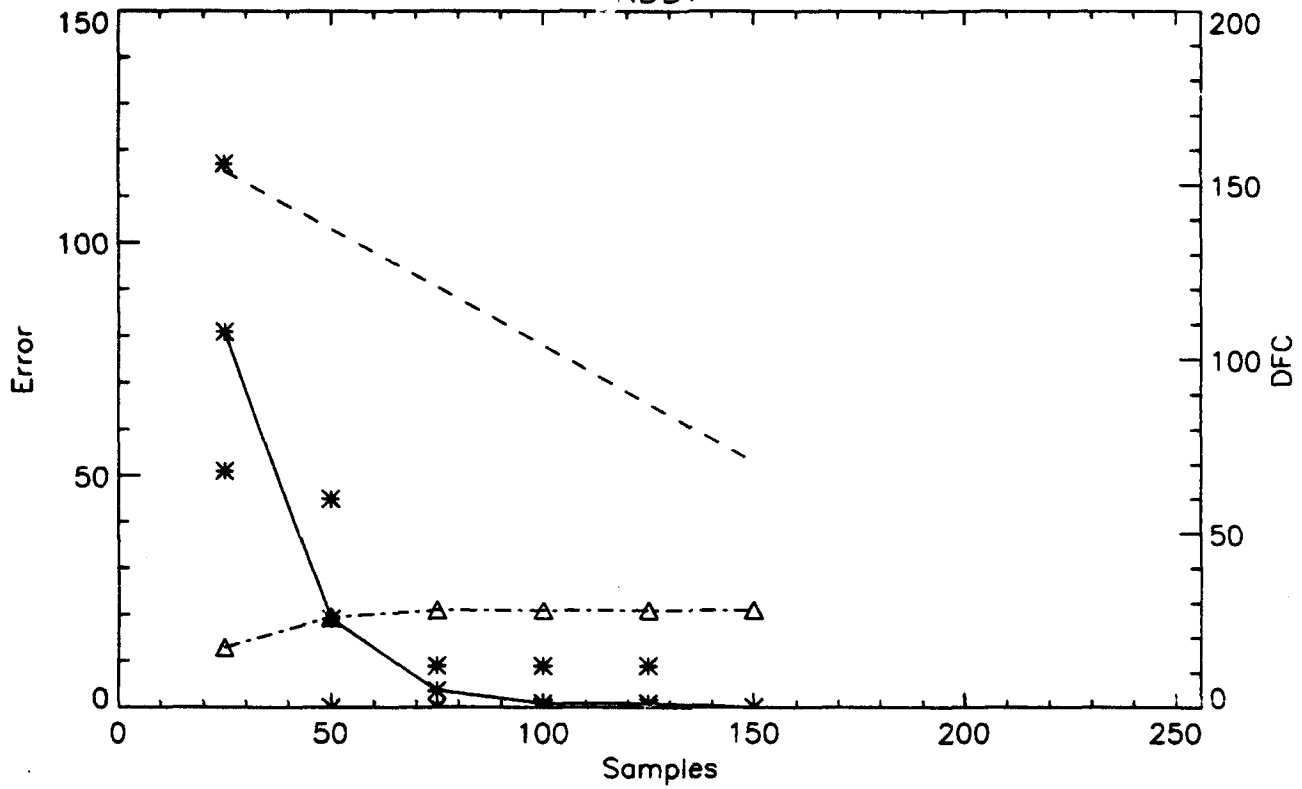
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

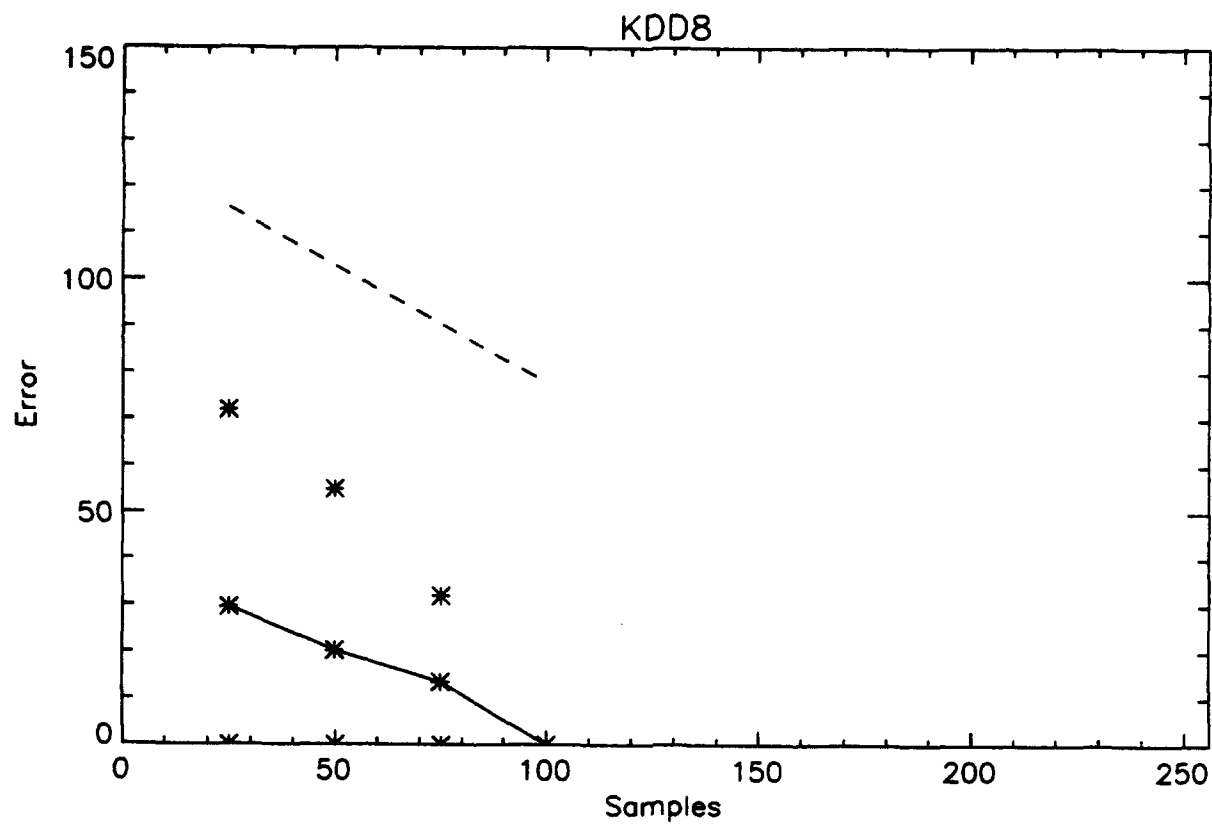
—*— Avg Error

KDD7



- Chance
- * Max error
- * Min error
- *— Avg error
- ...◊... Don't cares
- Δ--- Avg DFC

FLASH dni0e300



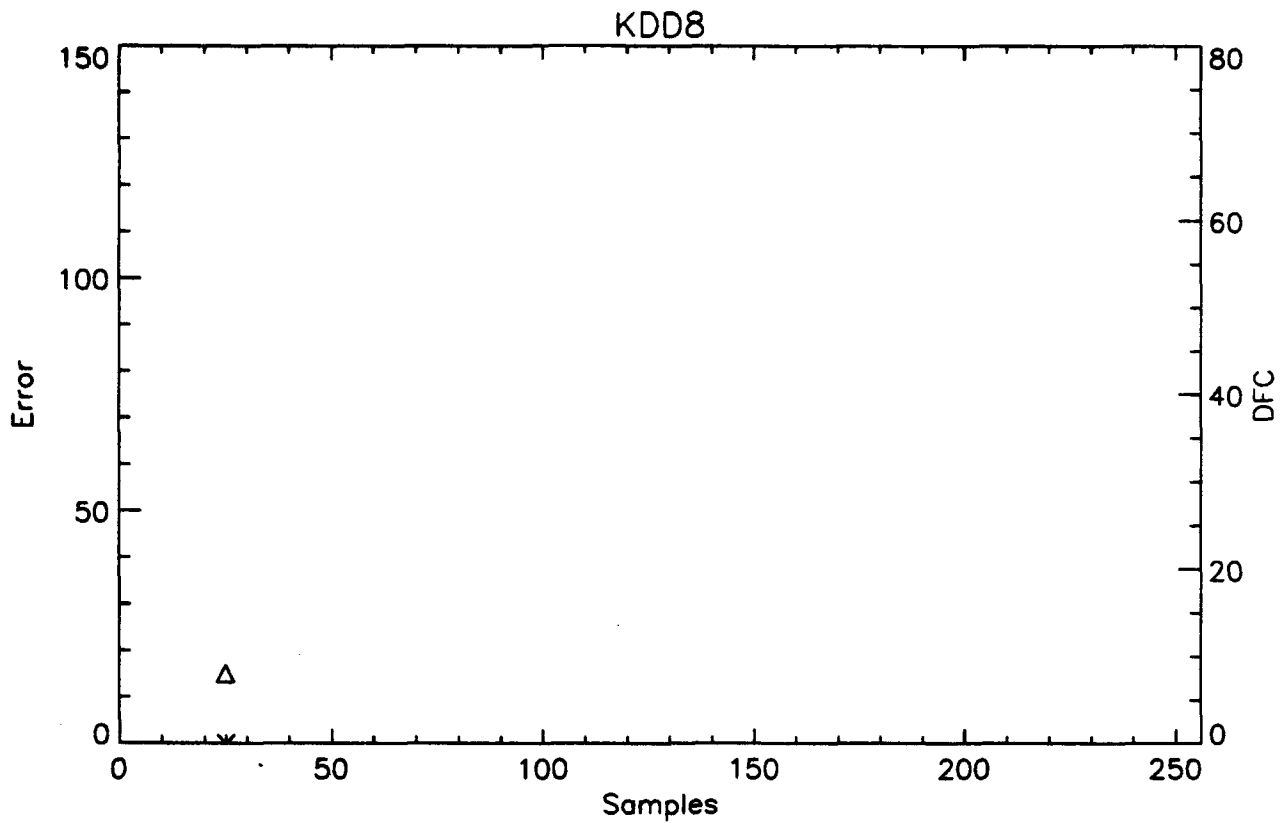
--- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

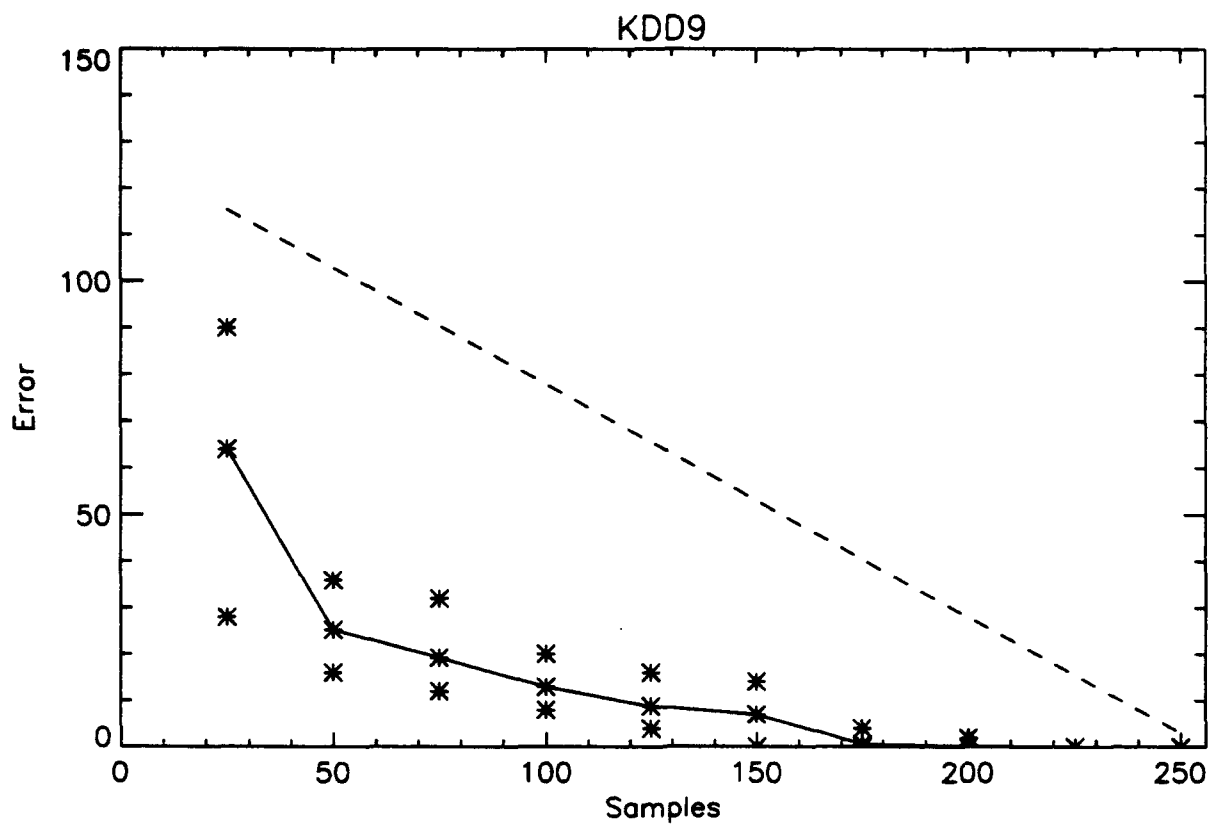
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



----- Chance

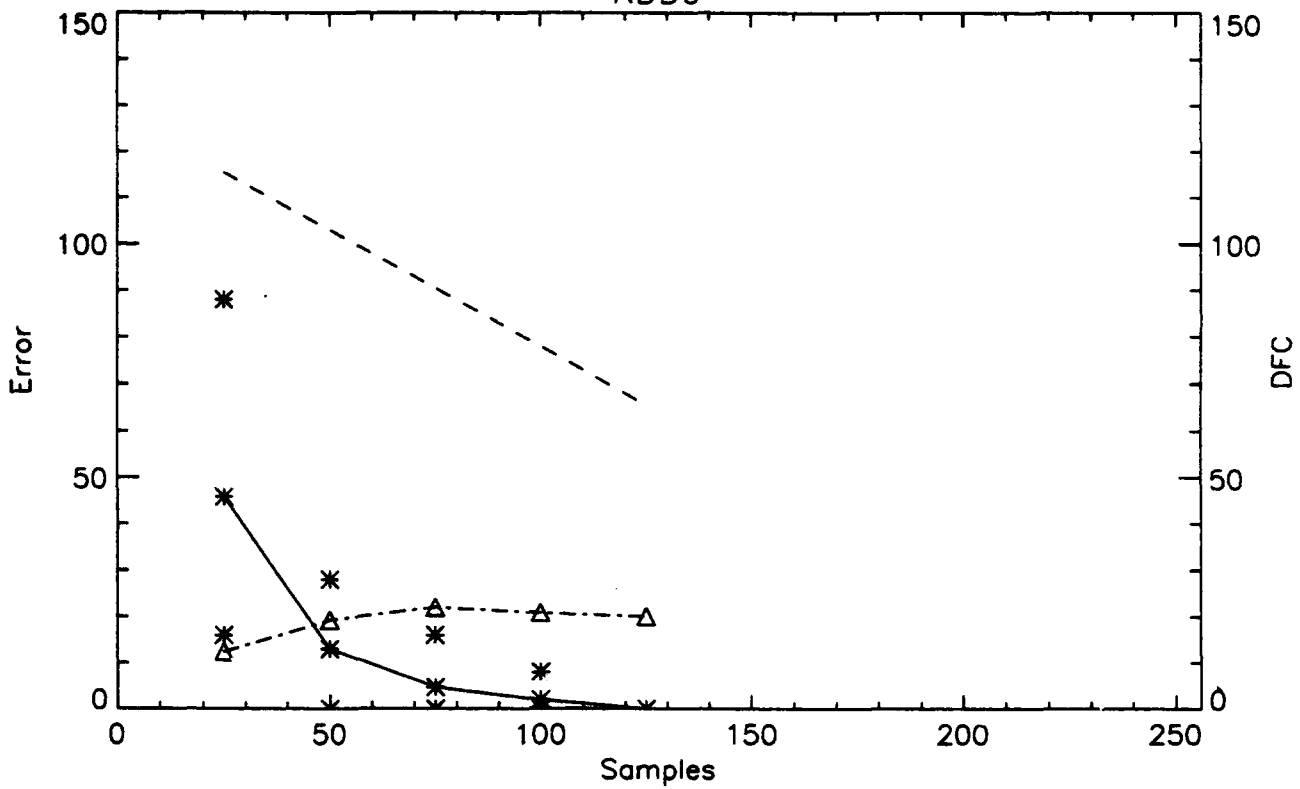
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

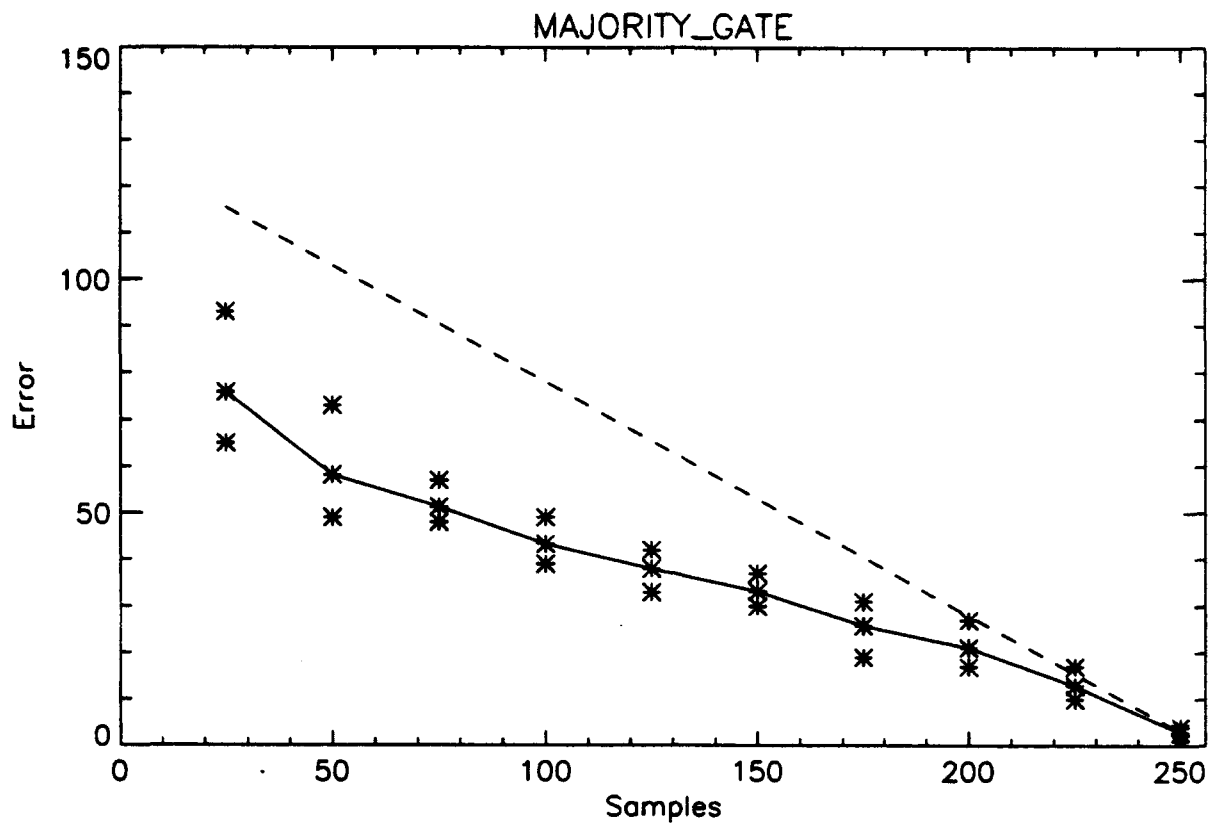
—*— Avg Error

KDD9



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



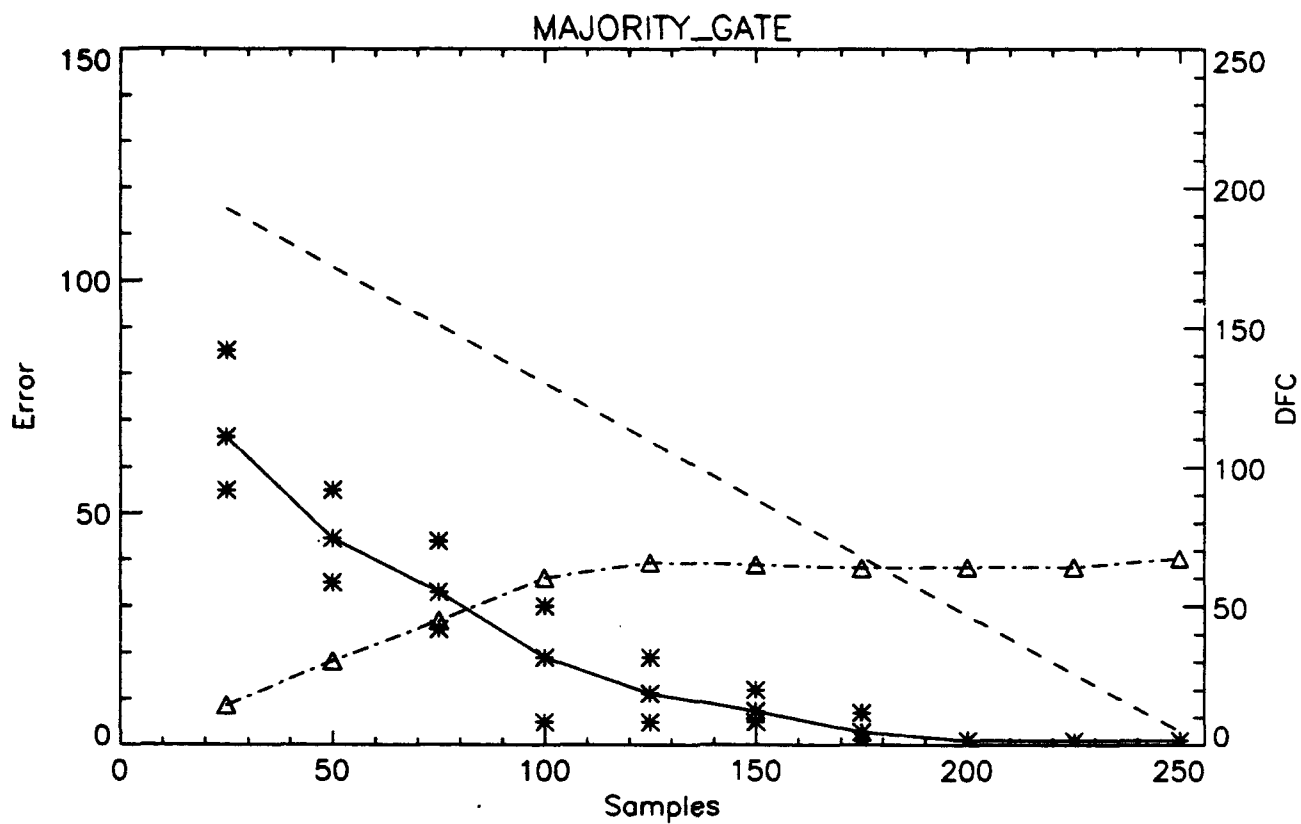
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

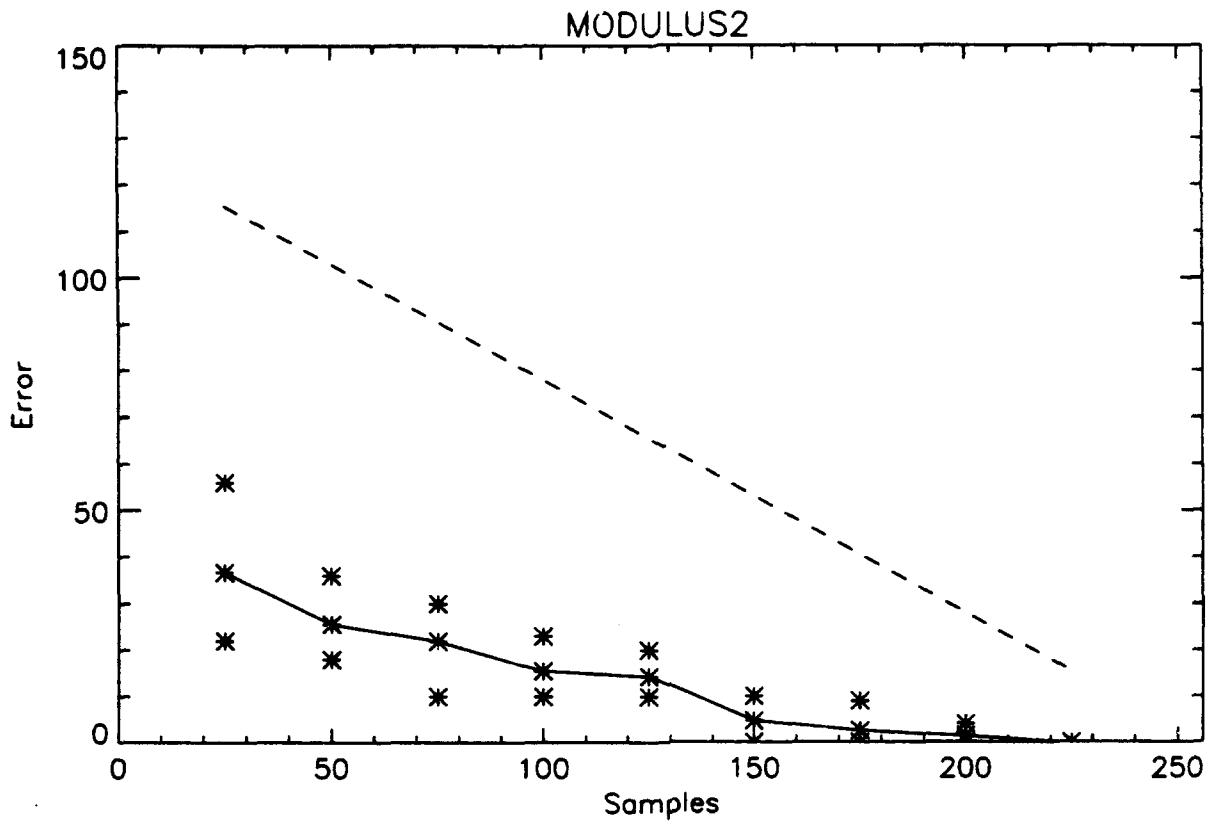
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-.-△-.-.- Avg DFC

FLASH dni0e300



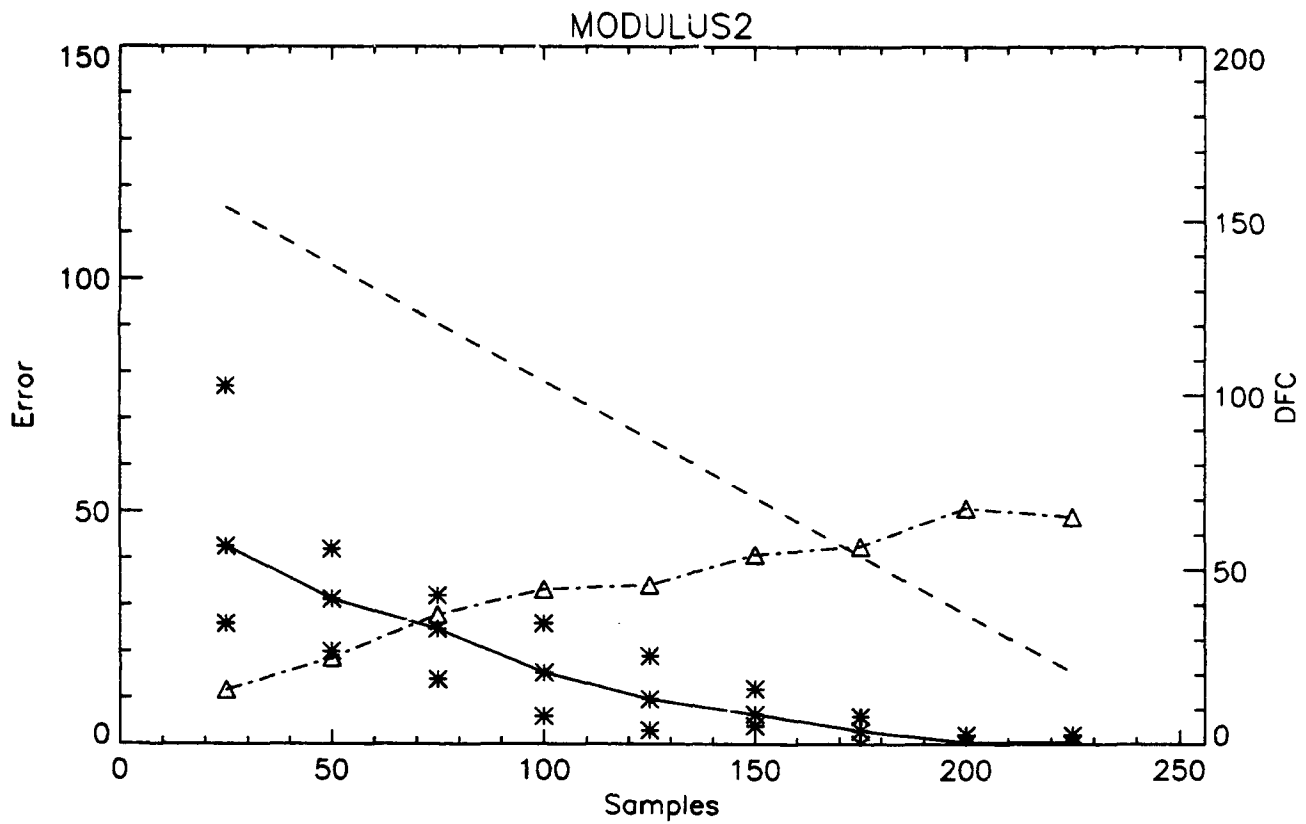
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

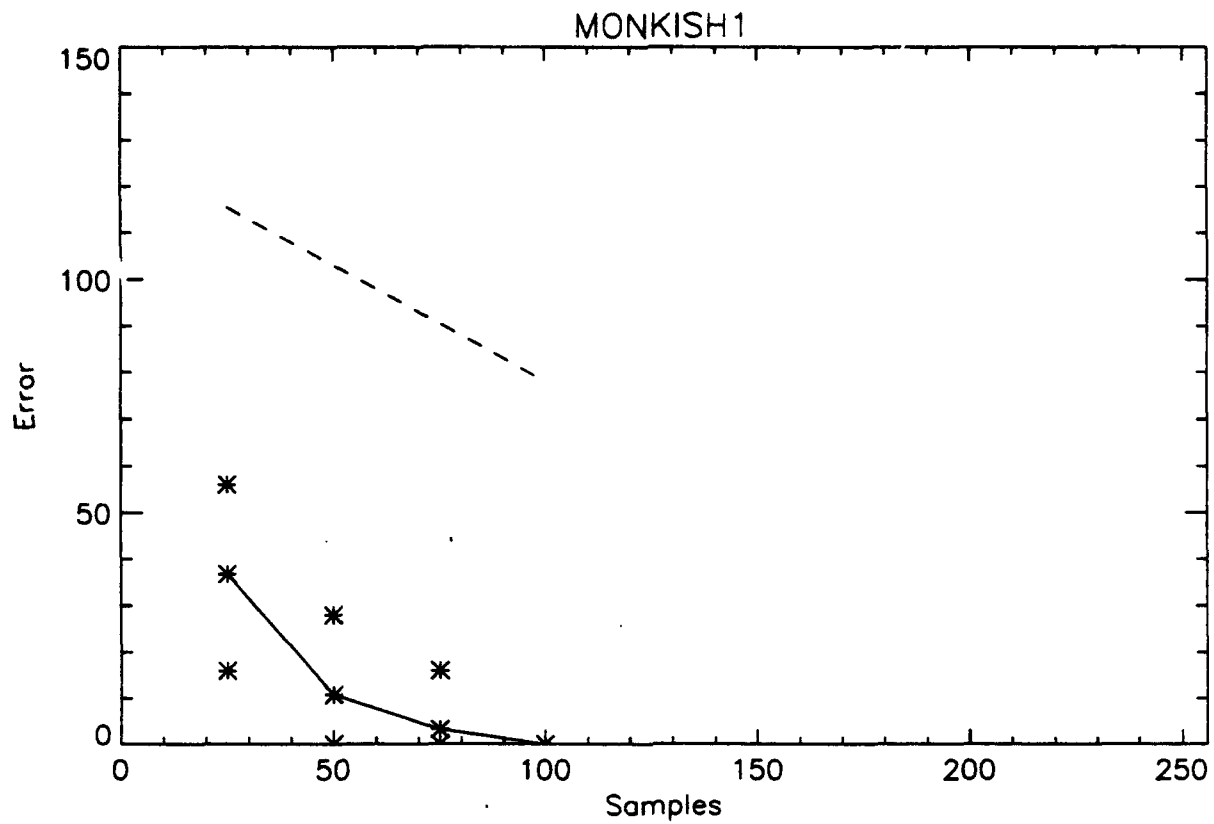
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



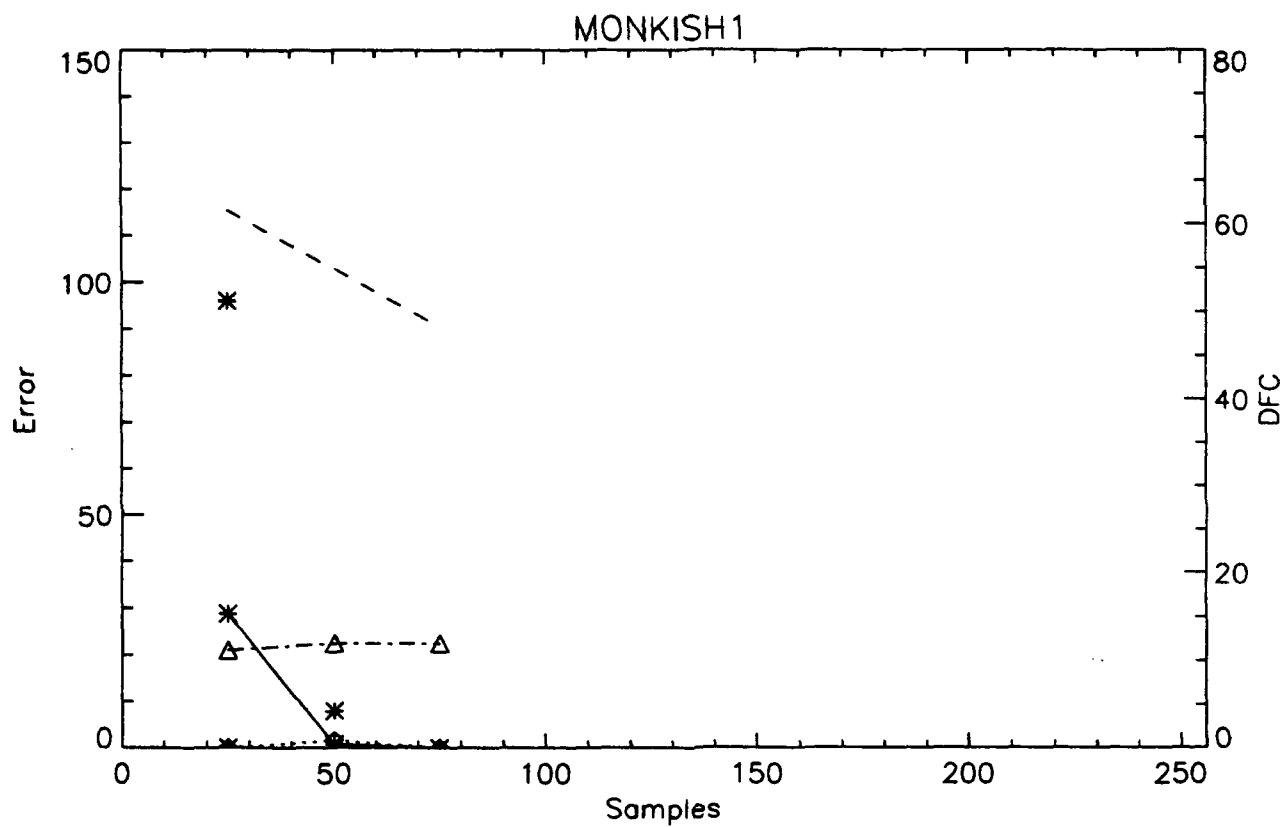
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

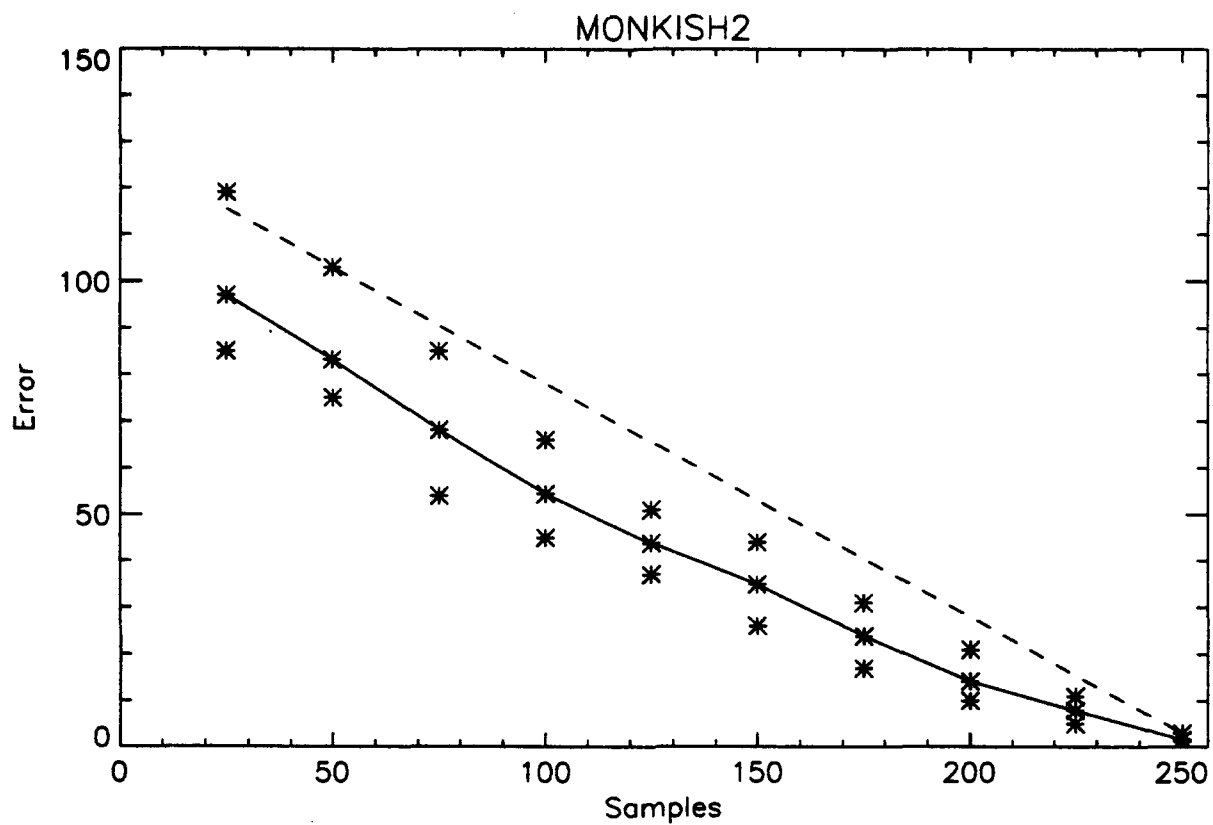
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



----- Chance

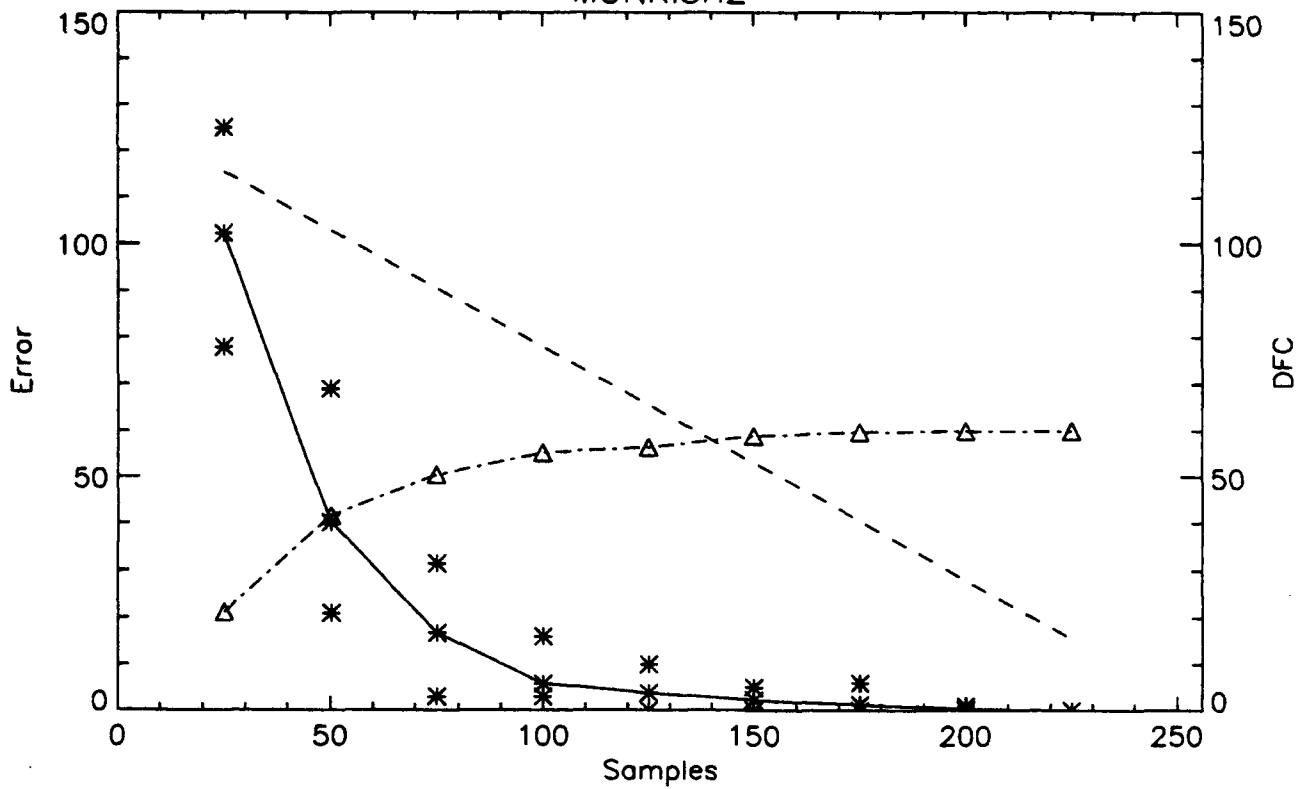
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

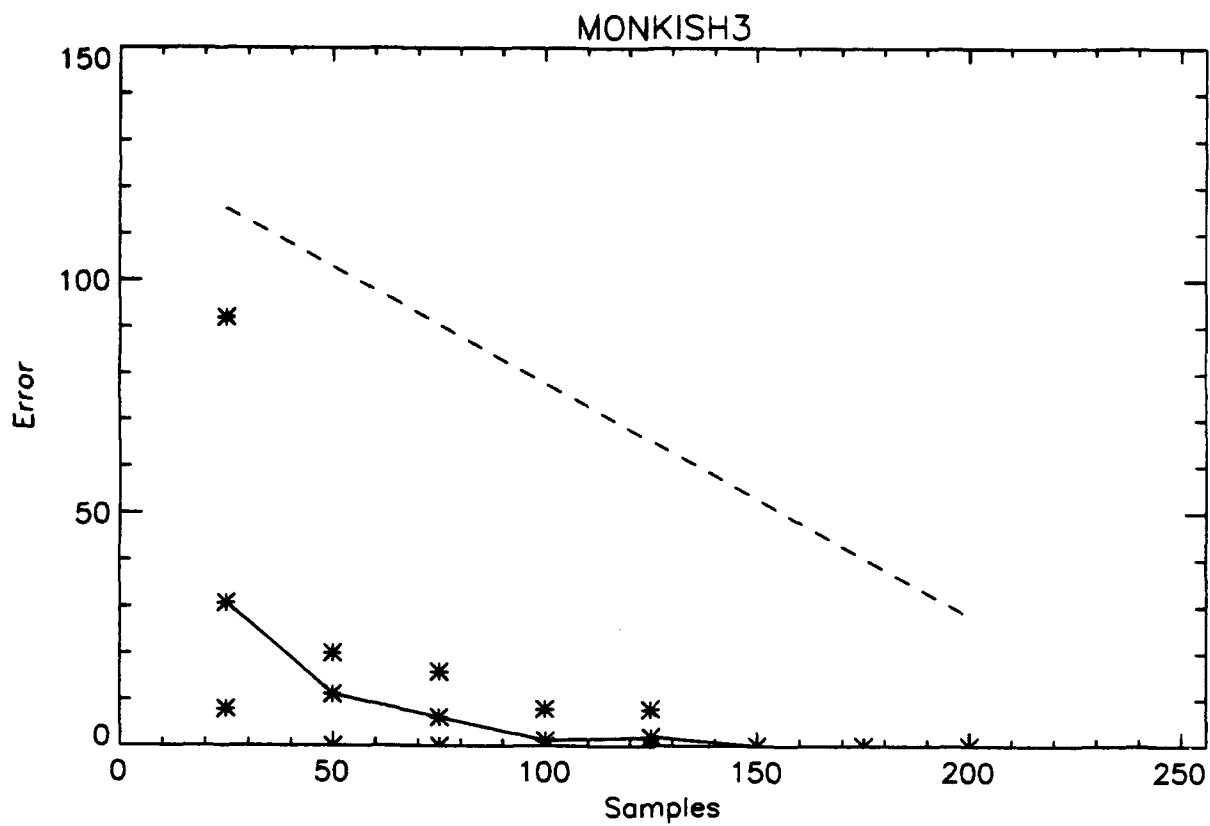
—*— Avg Error

MONKISH2



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



----- Chance

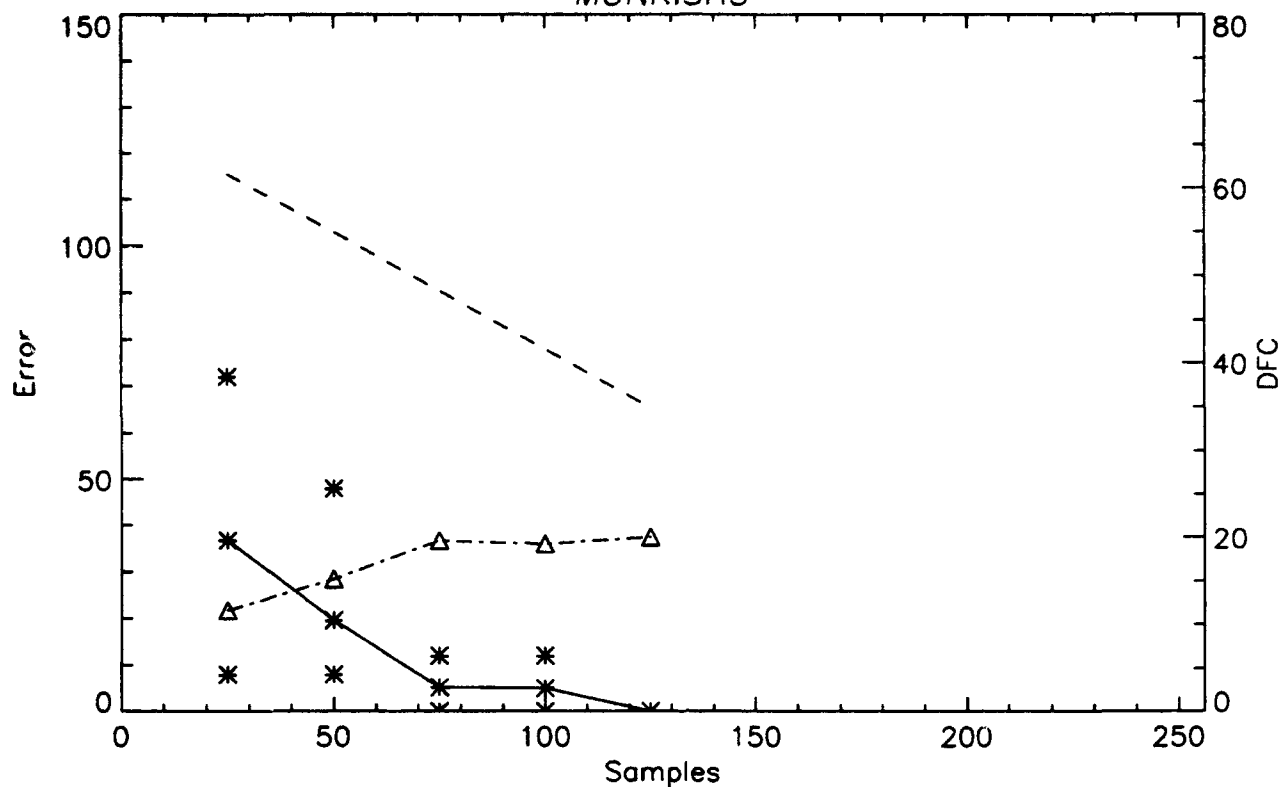
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

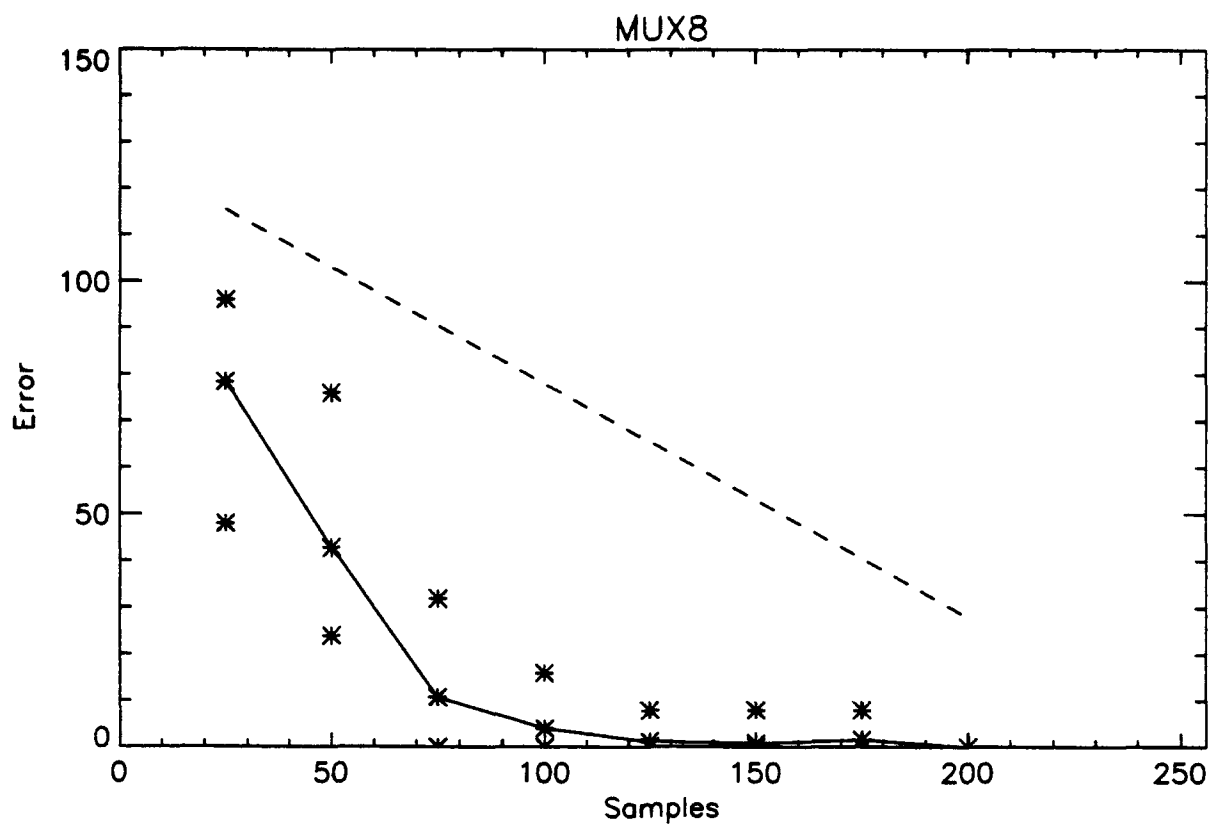
—*— Avg Error

MONKISH3



- Chance
- * Max error
- * Min error
- *— Avg error
- ...◇... Don't cares
- △--- Avg DFC

FLASH dni0e300



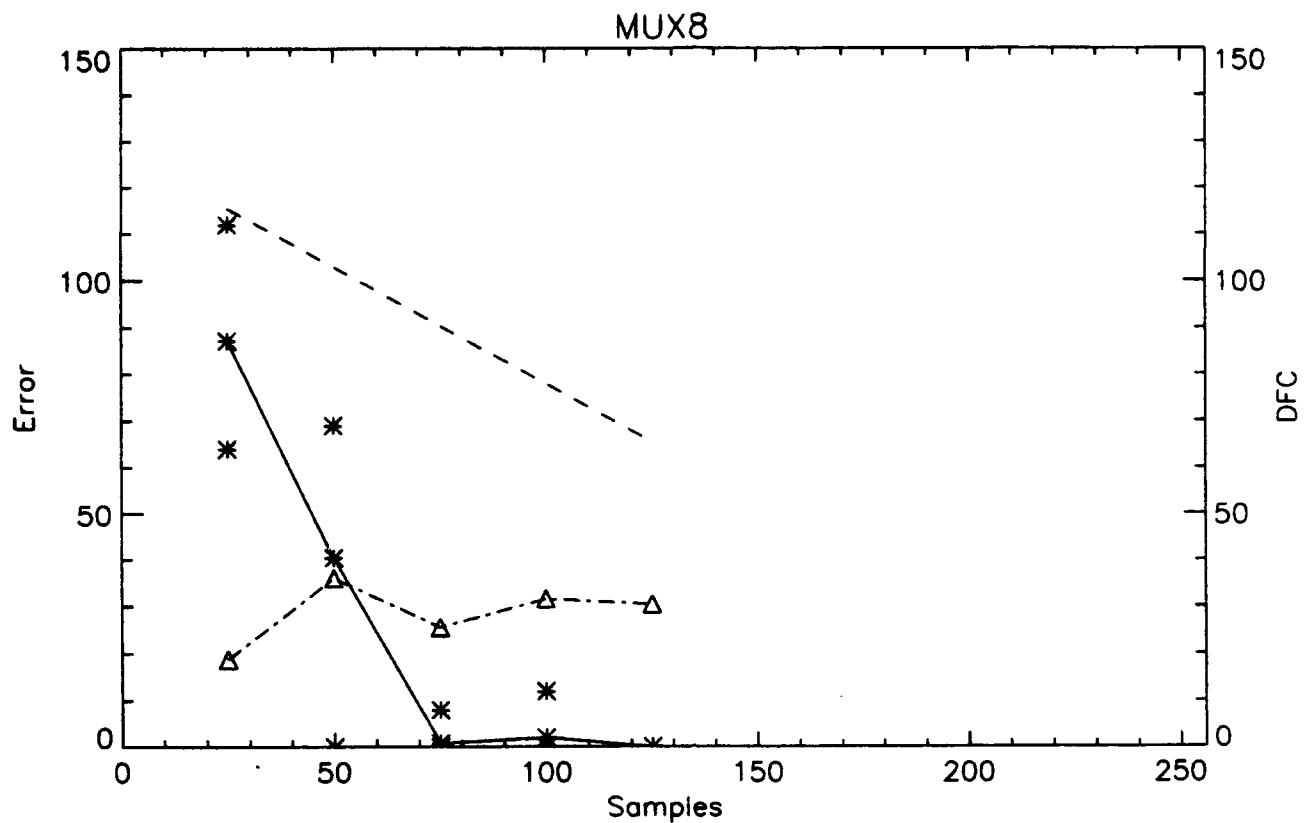
--- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

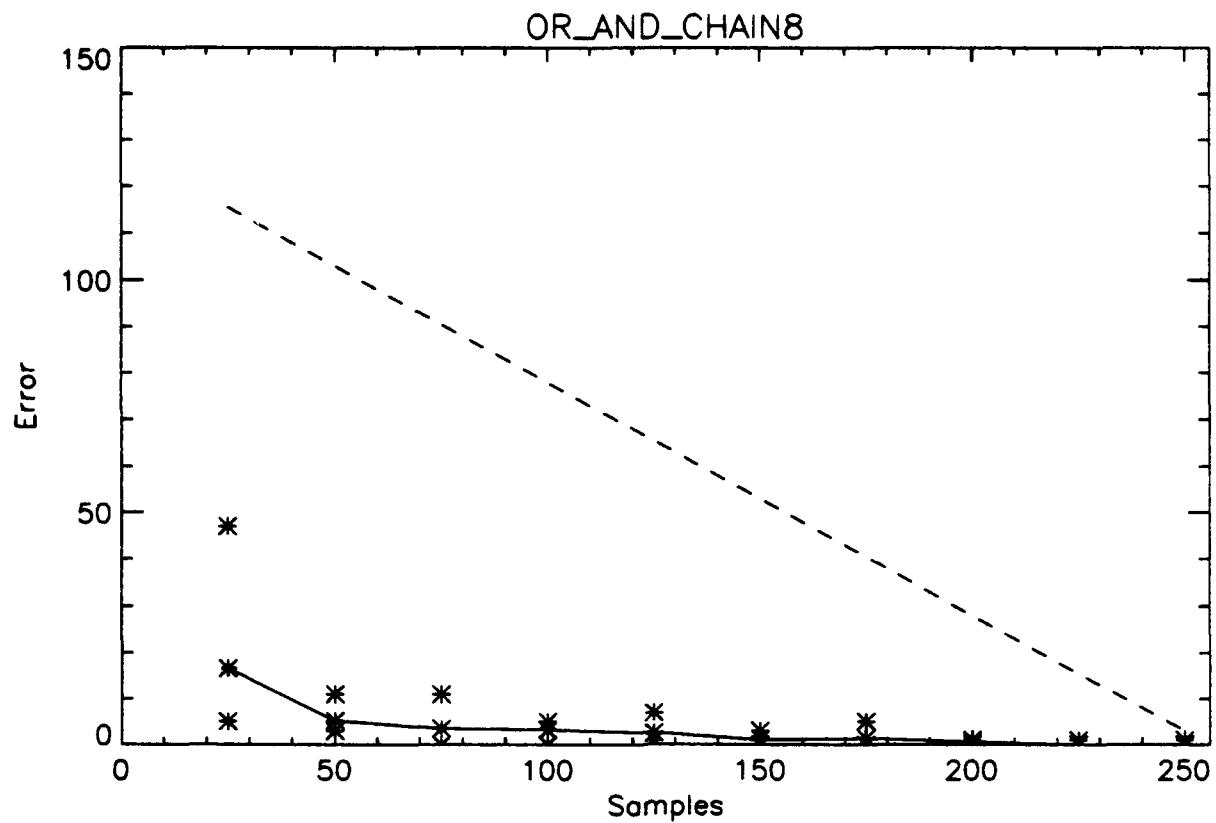
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



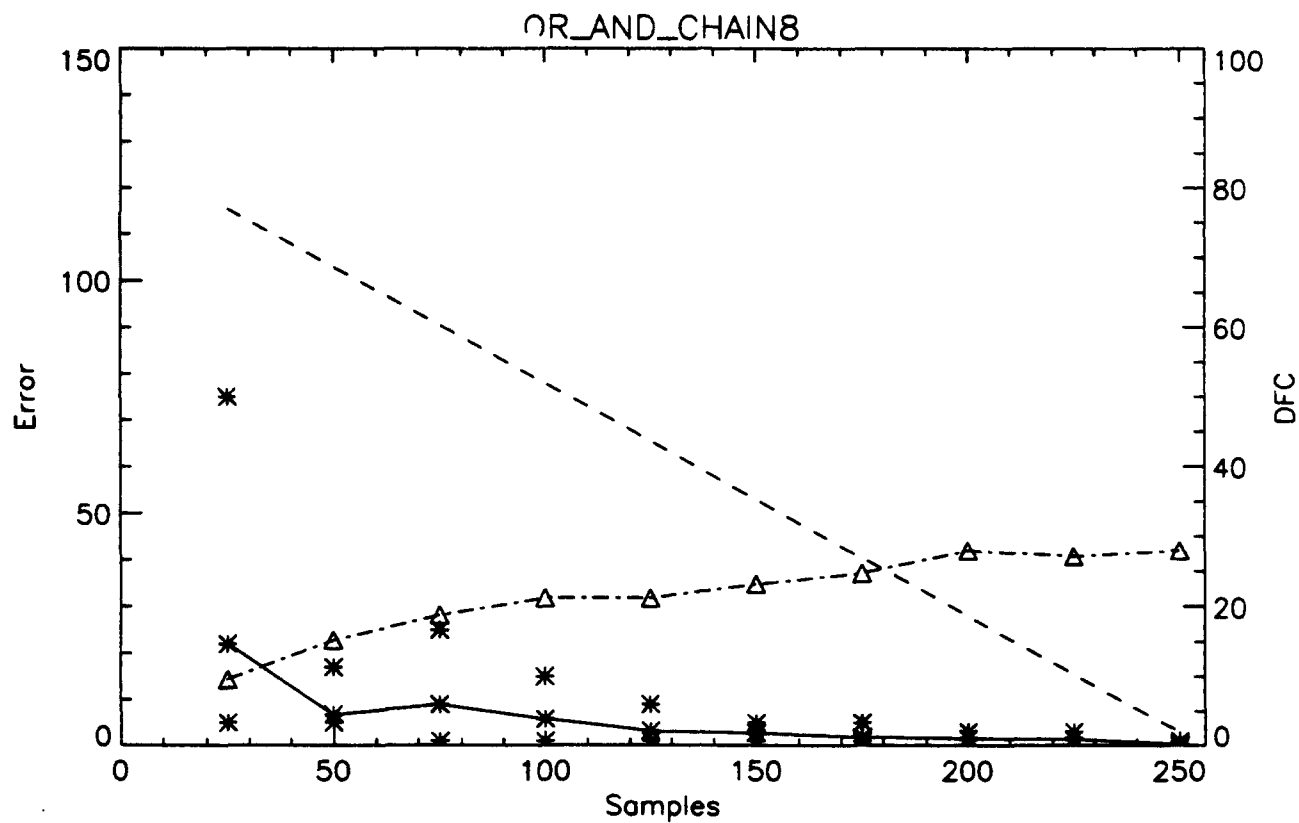
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

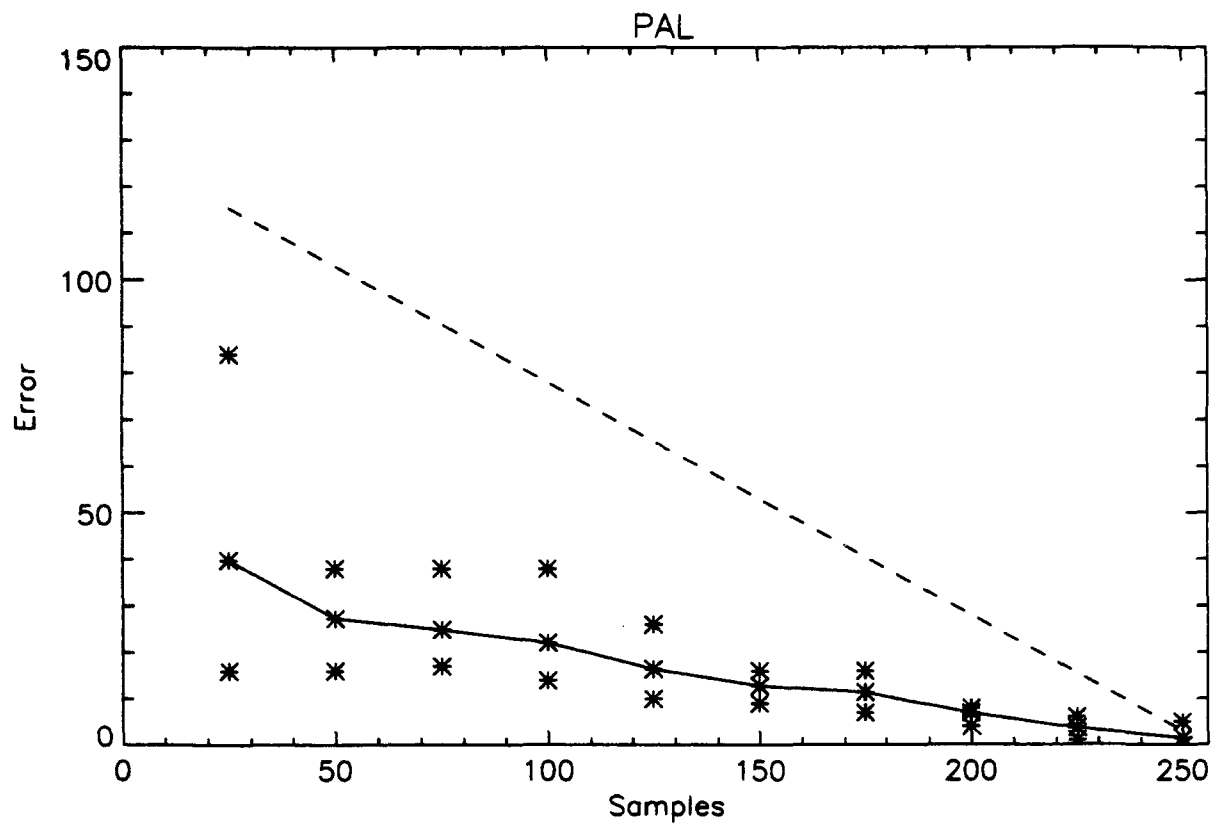
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



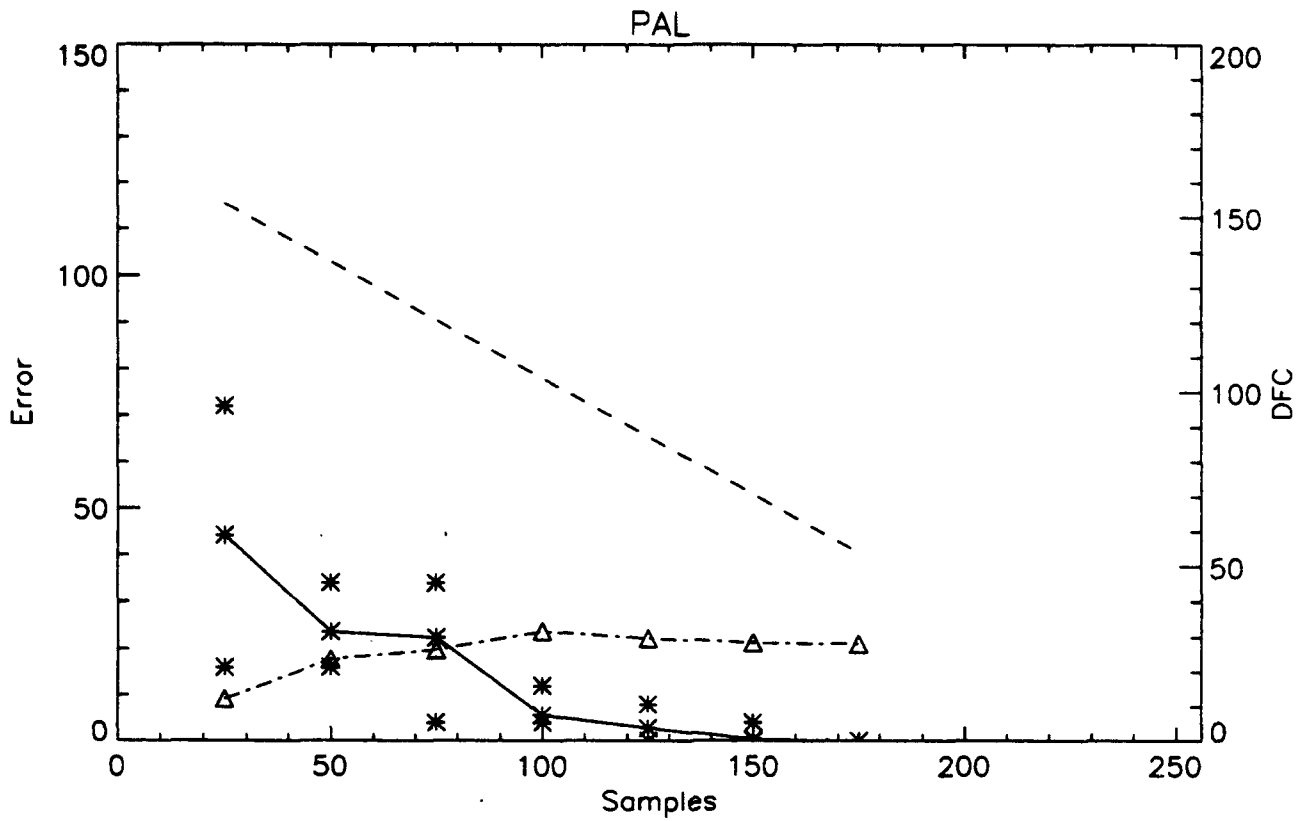
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

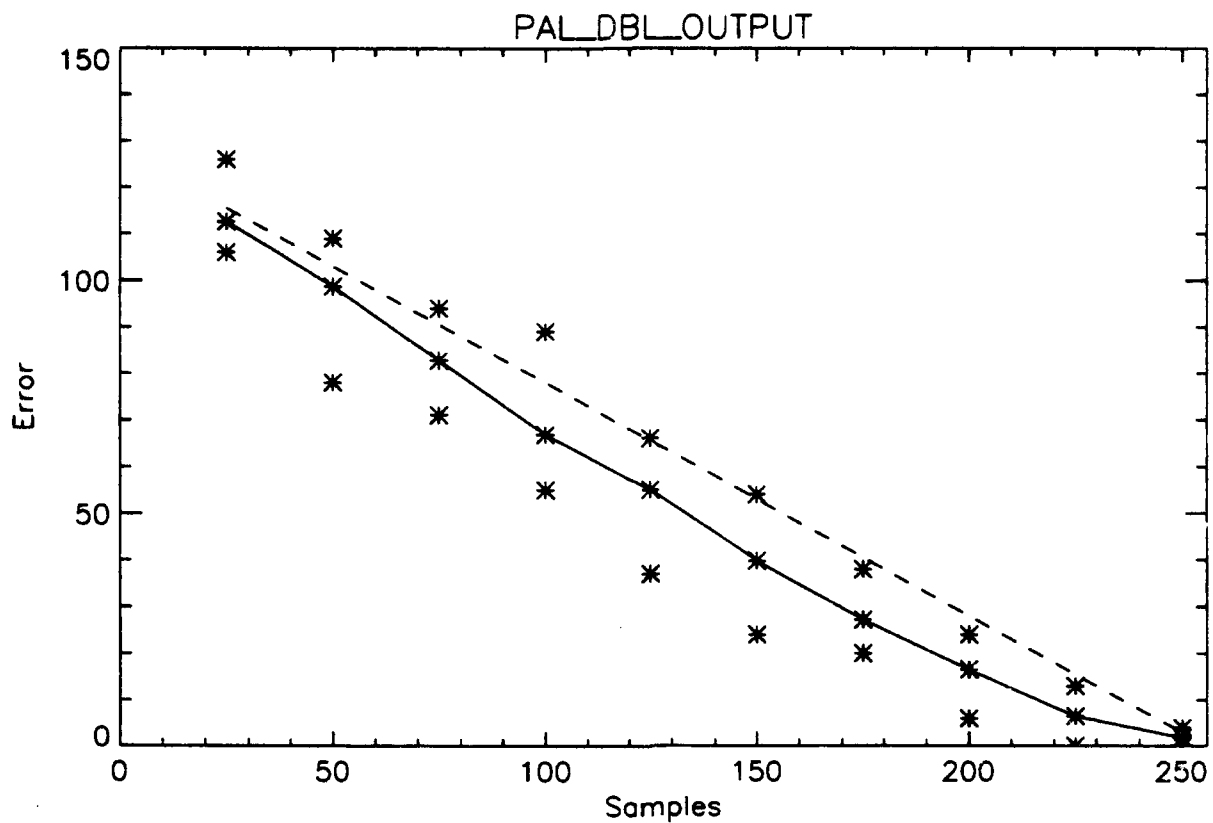
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300

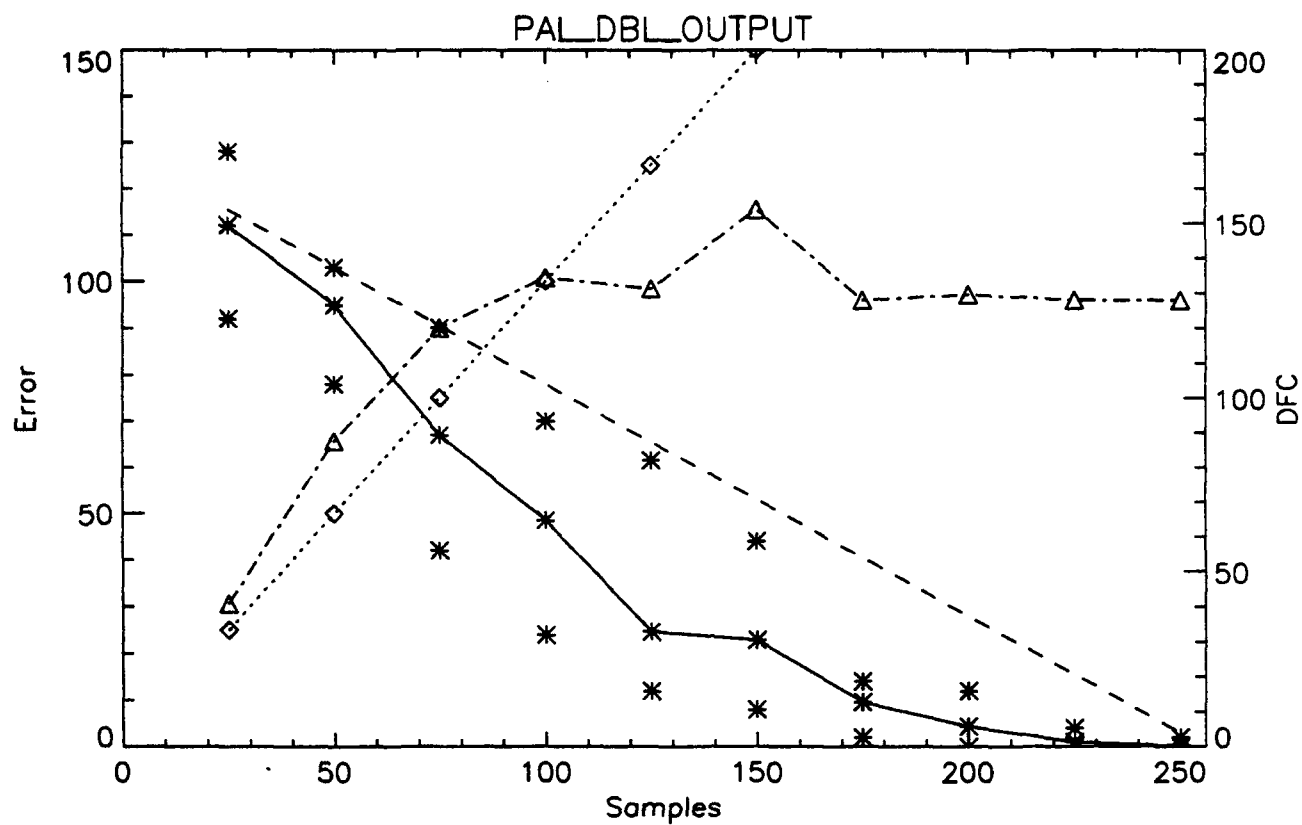


C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

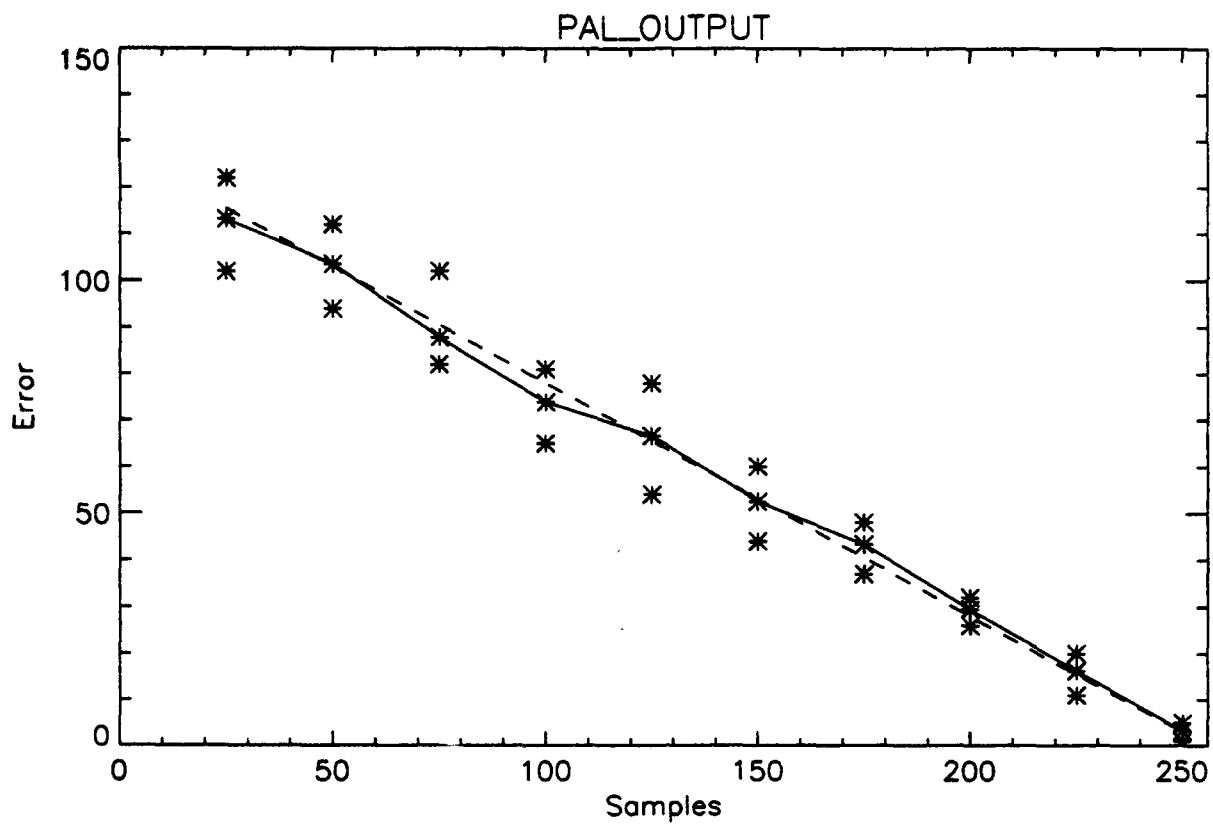
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - -△- - - Avg DFC

FLASH dni0e300



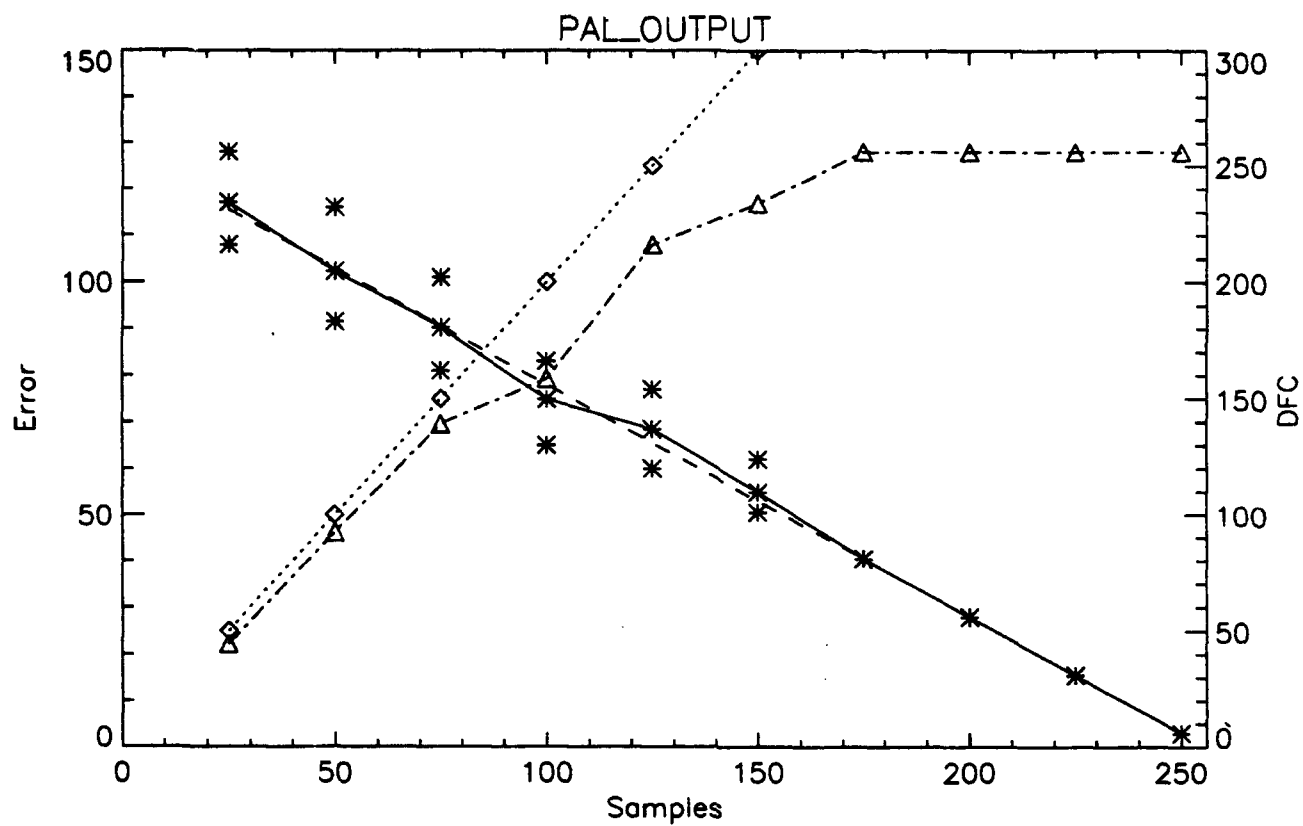
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

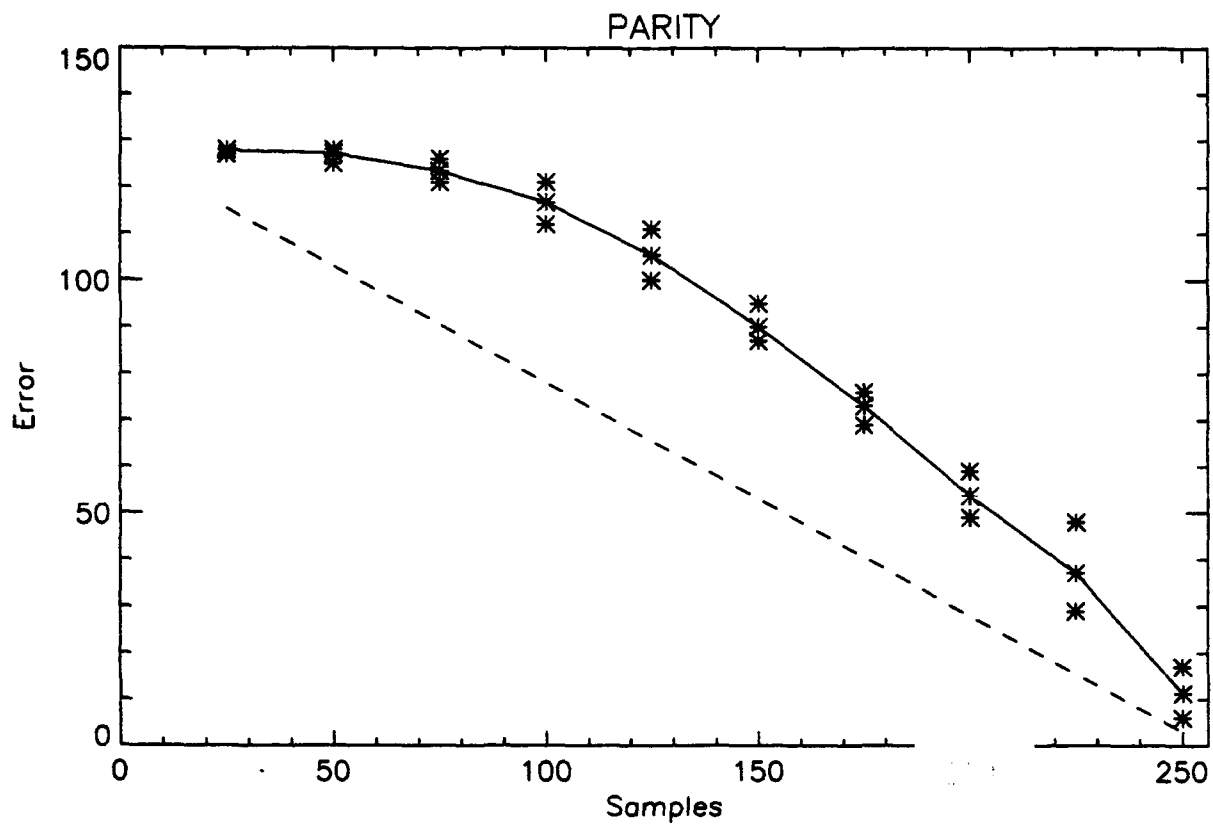
* Min Error

—*— Avg Error



- Chance
- * Max error
- *. Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



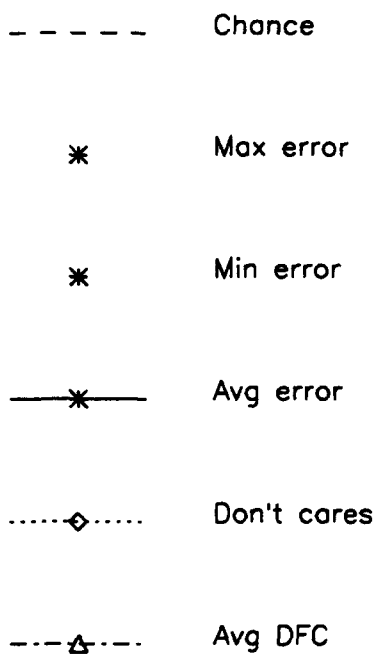
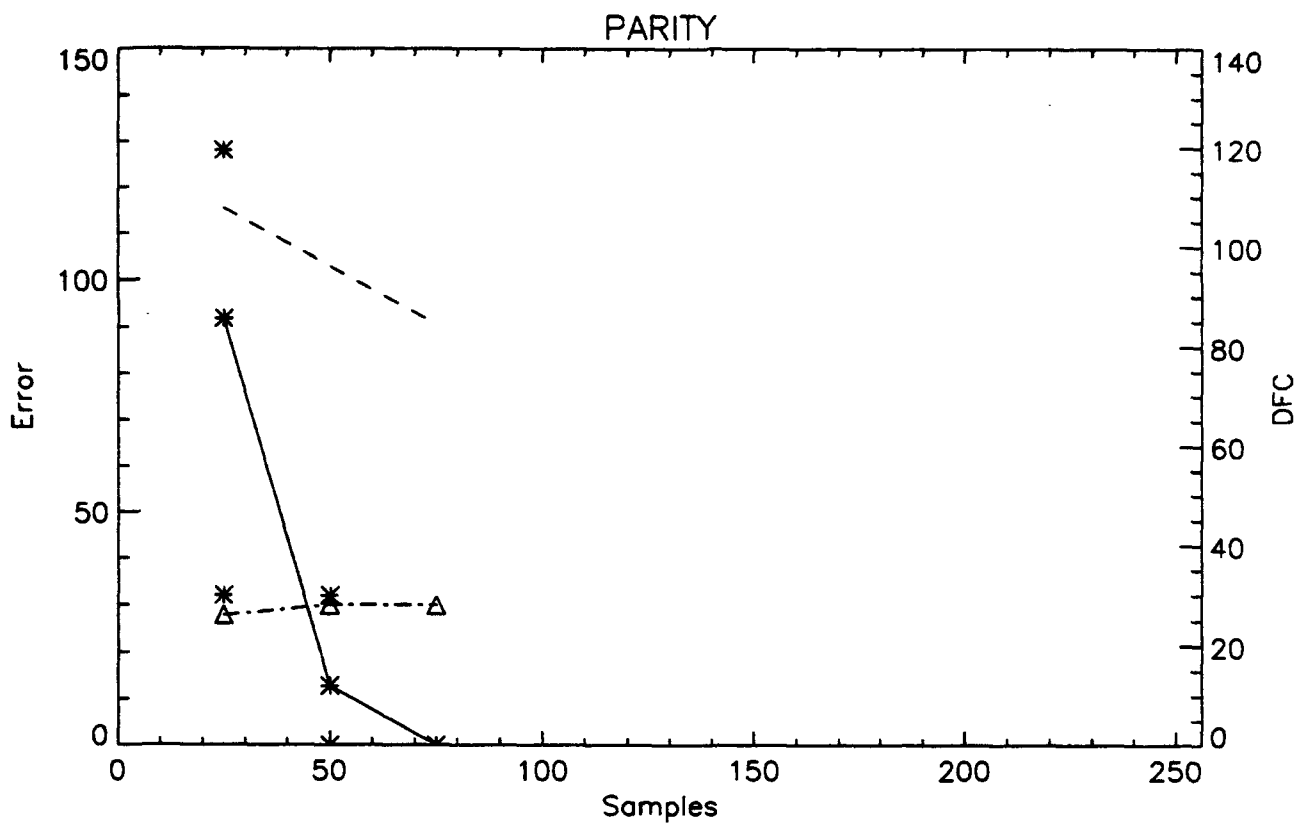
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

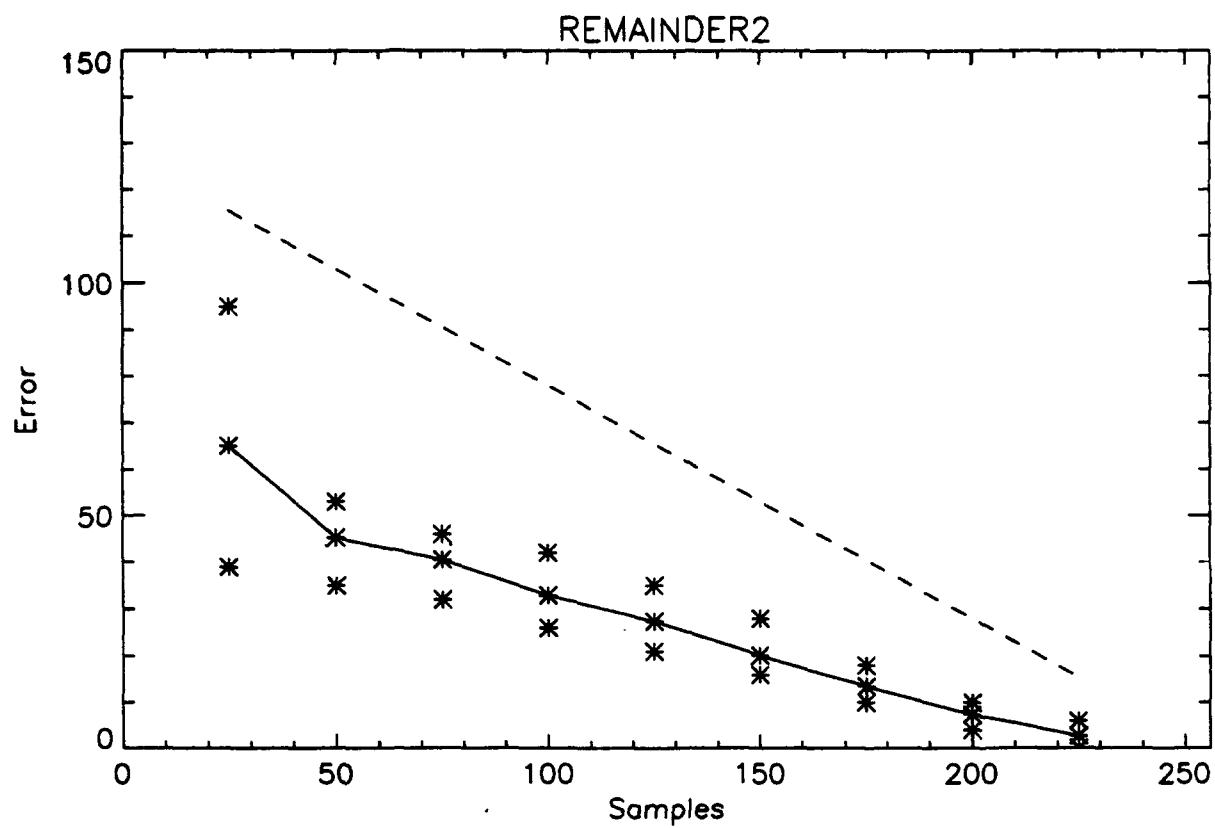
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



----- Chance

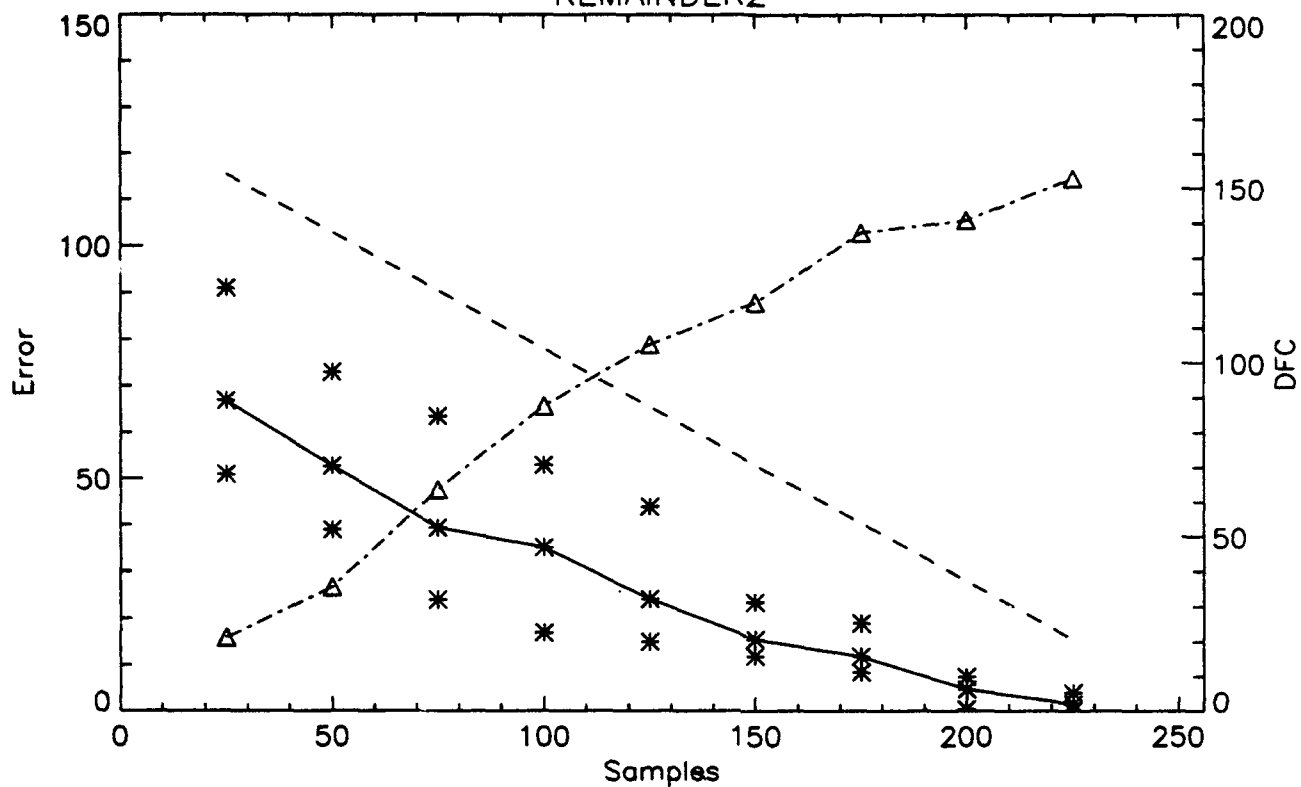
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

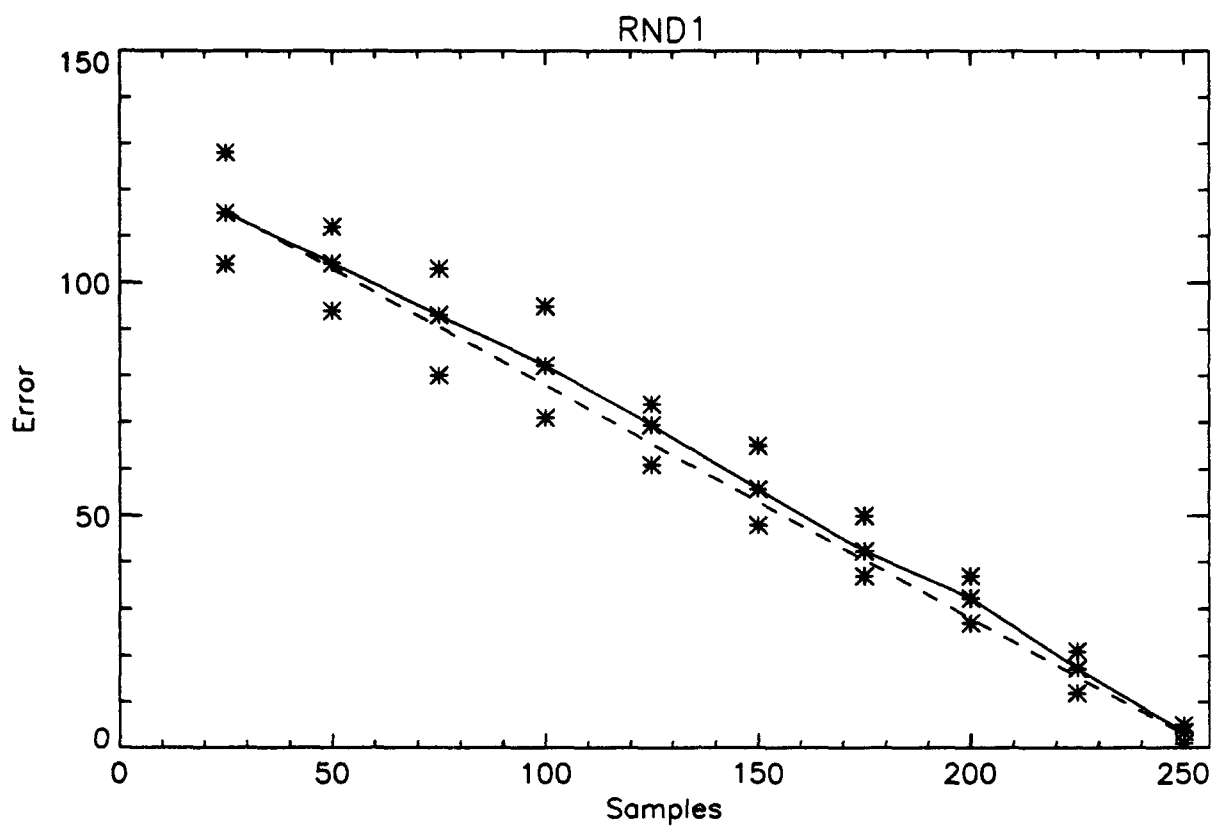
—*— Avg Error

REMAINDER2



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



----- Chance

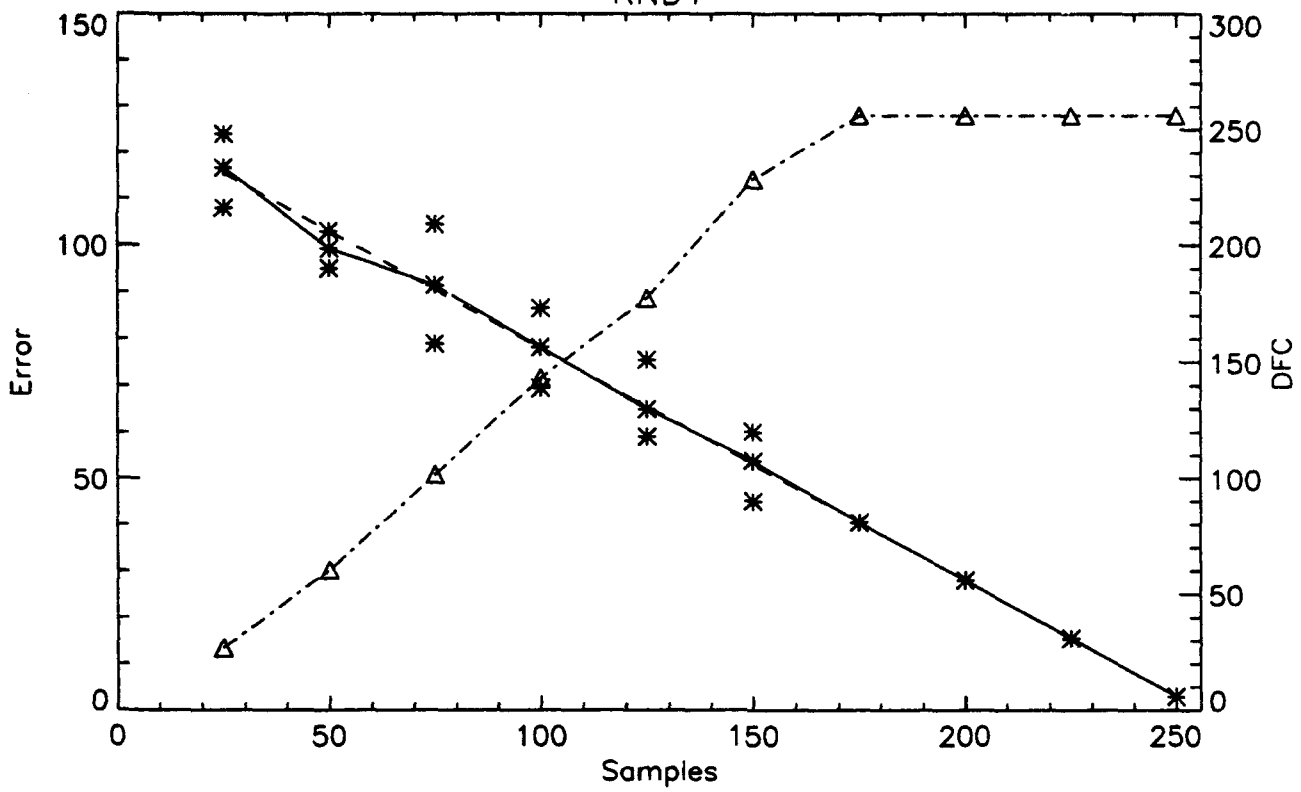
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

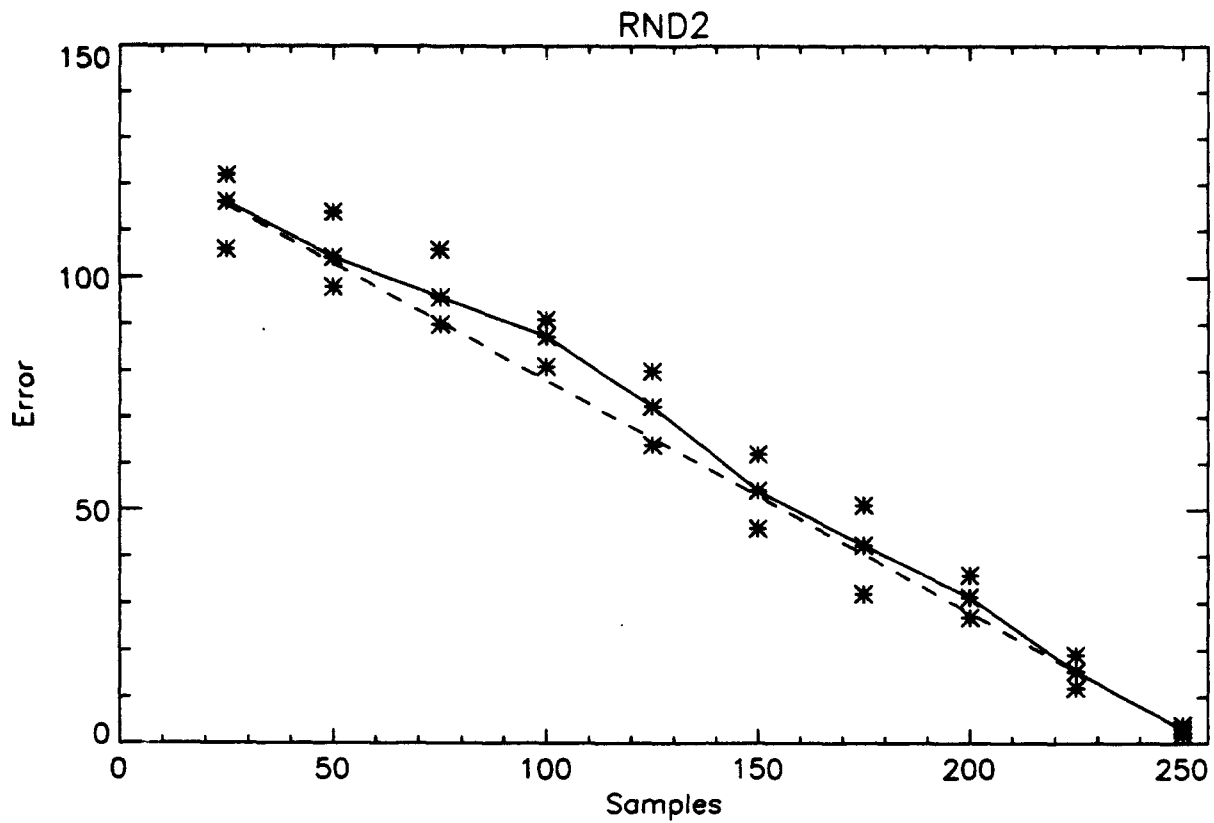
—*— Avg Error

RND1



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - Δ - - - Avg DFC

FLASH dni0e300



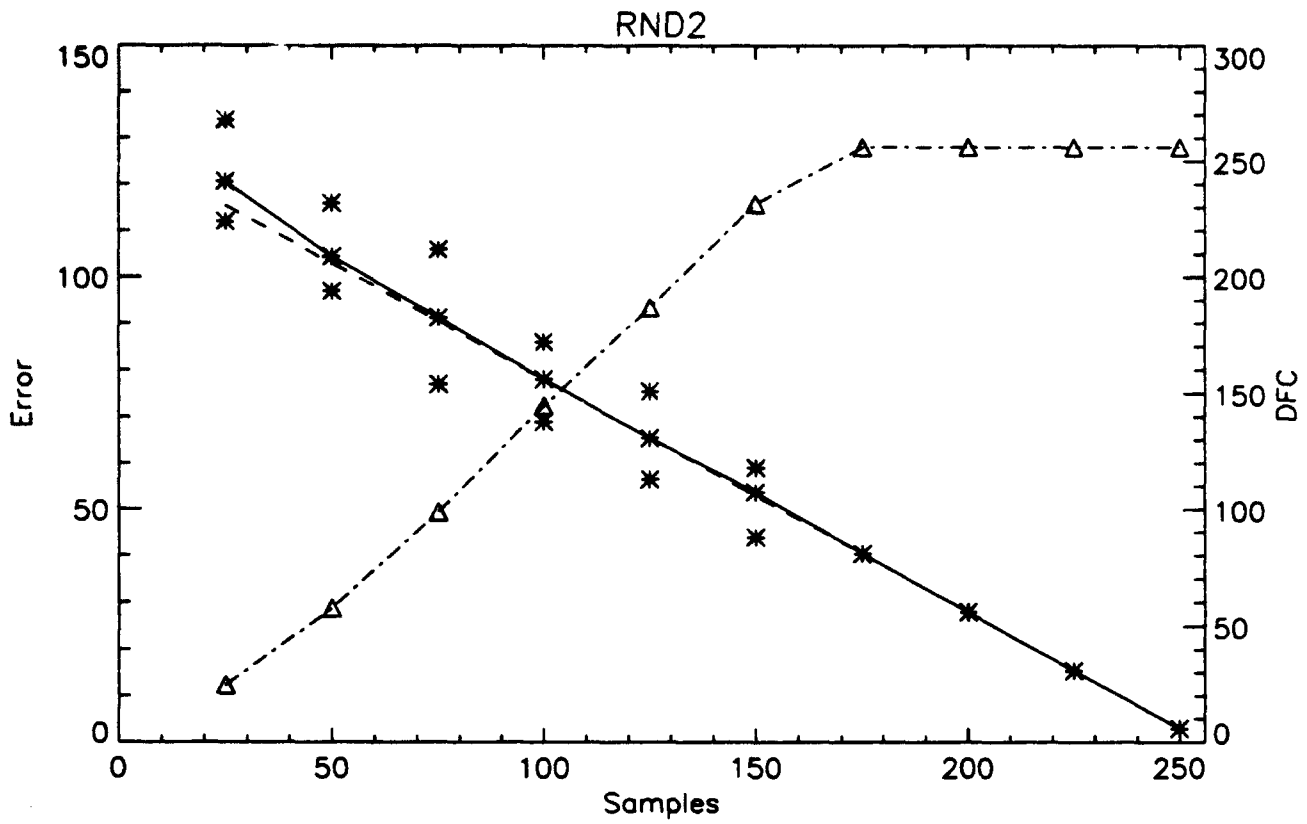
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

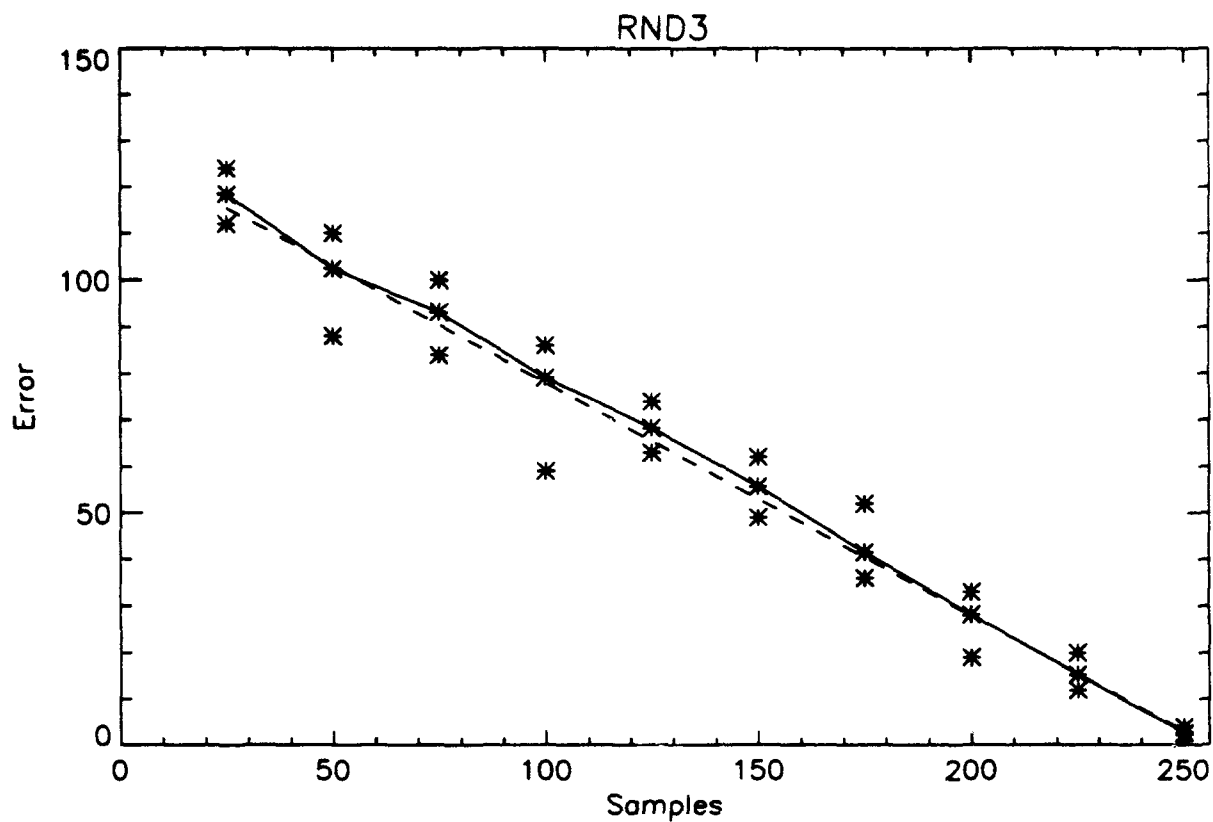
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



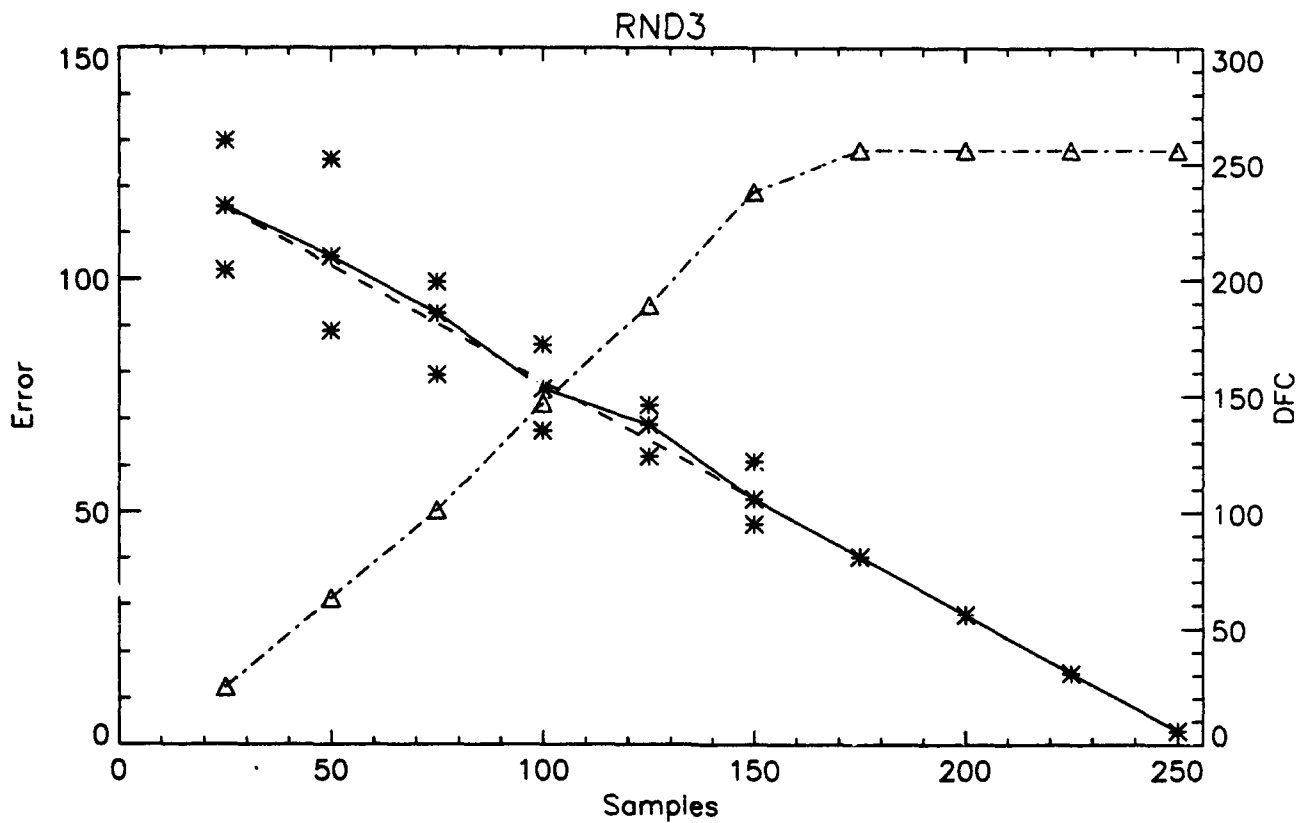
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

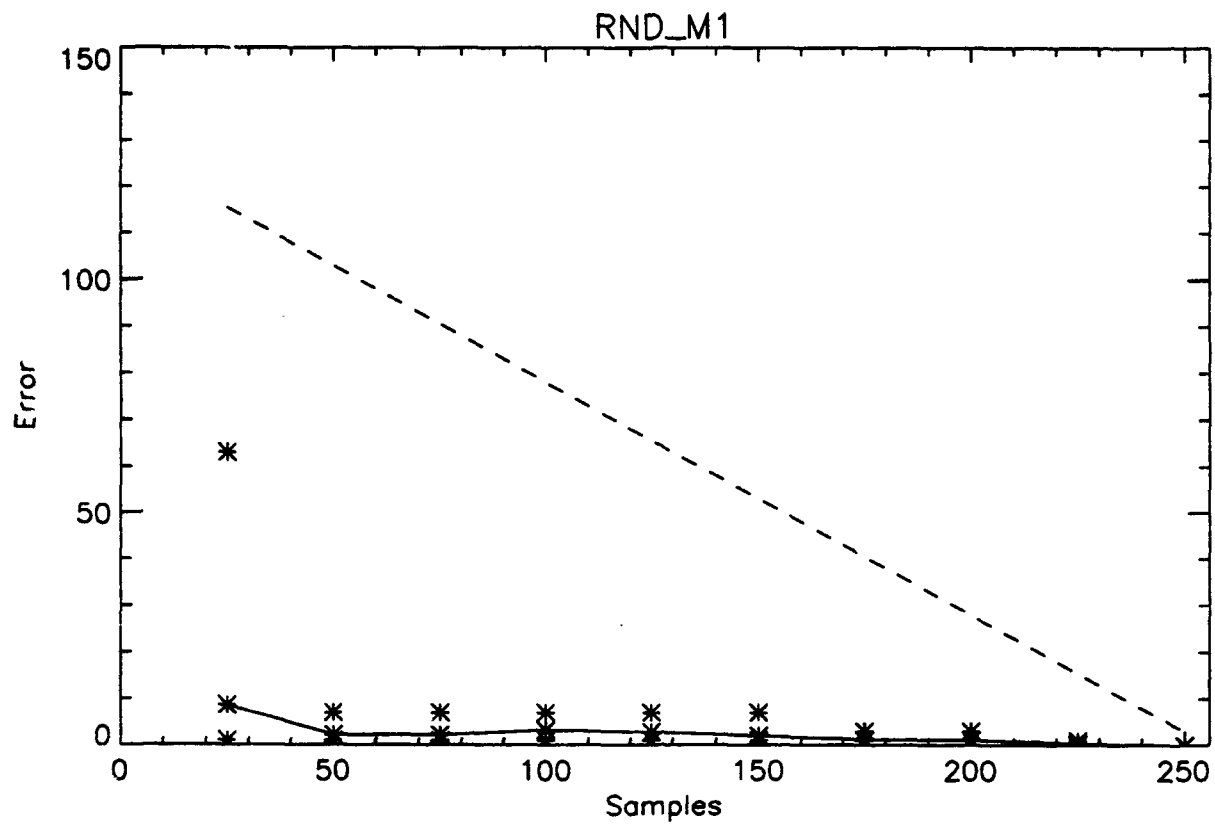
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



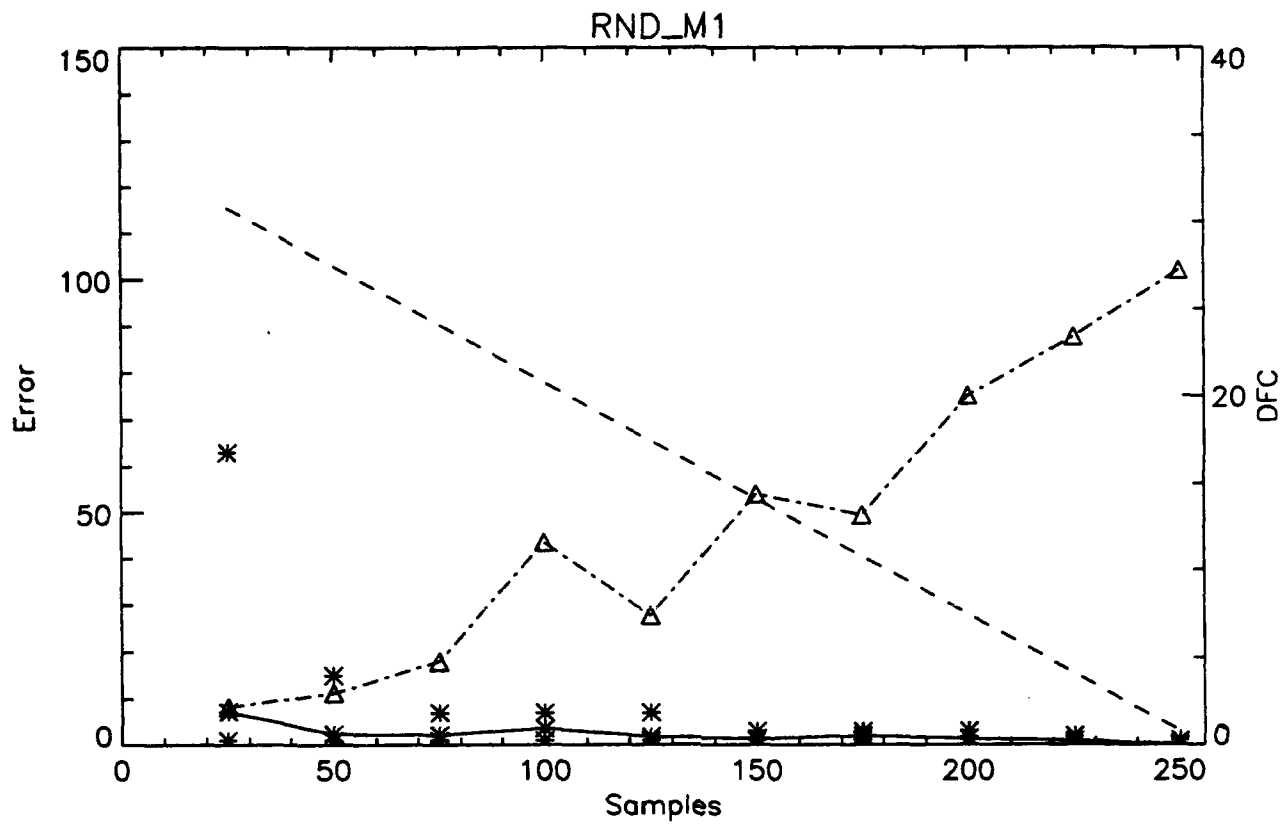
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

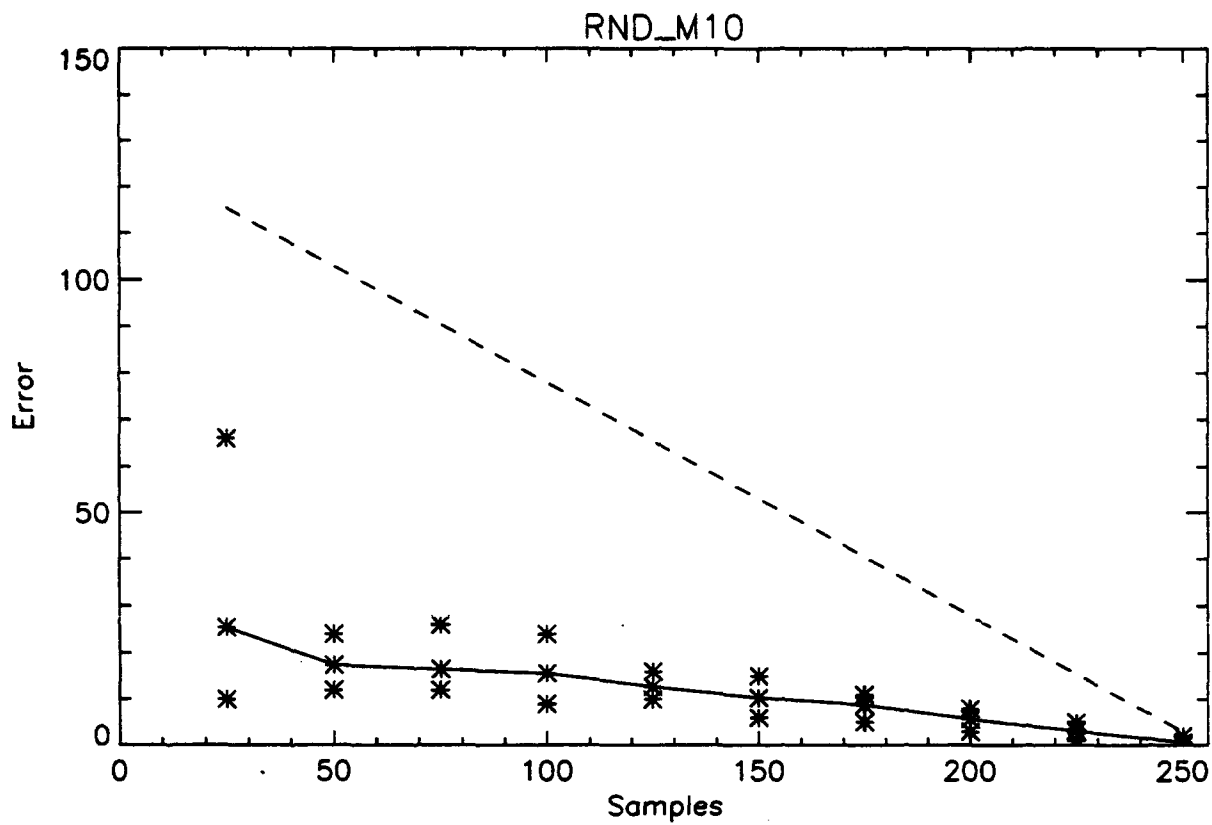
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *--- Avg error
-◇..... Don't cares
- .-△-.- Avg DFC

FLASH dni0e300



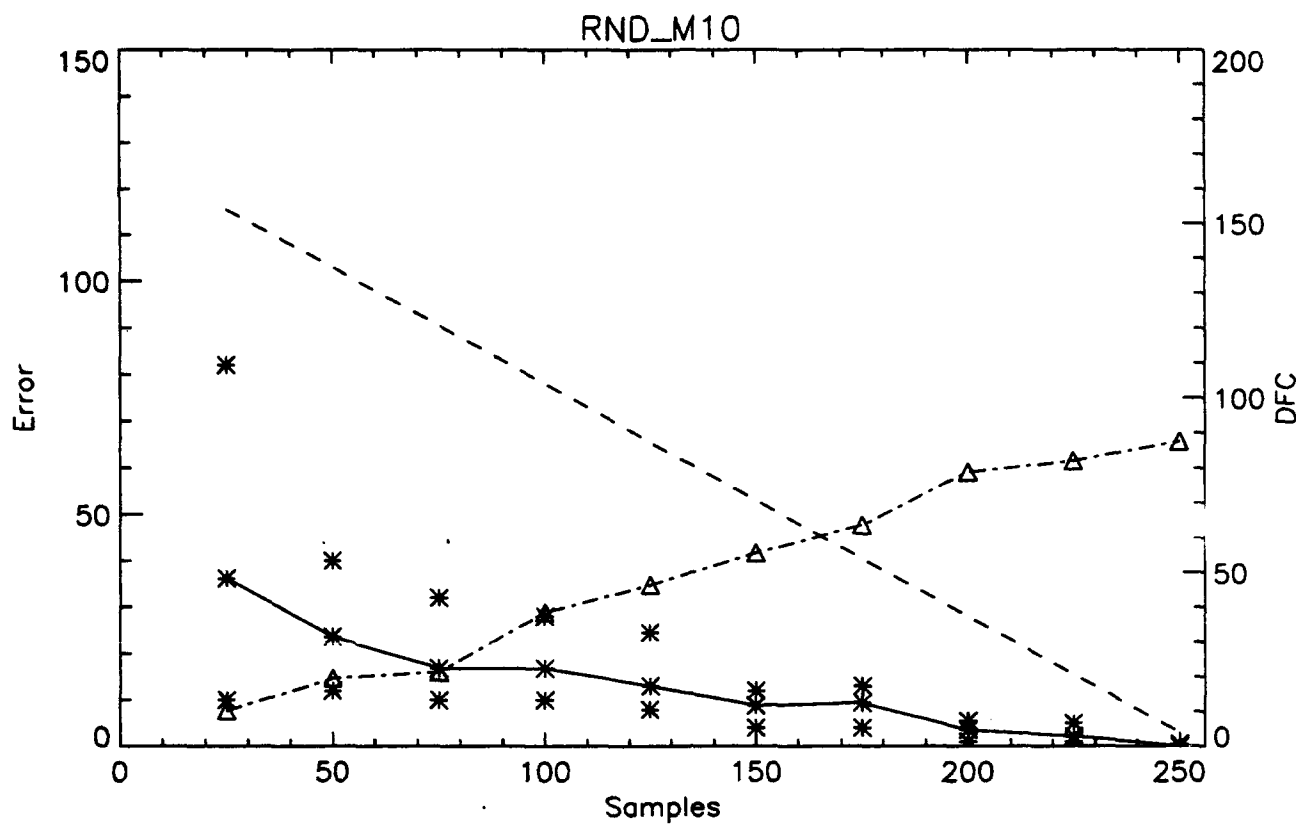
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

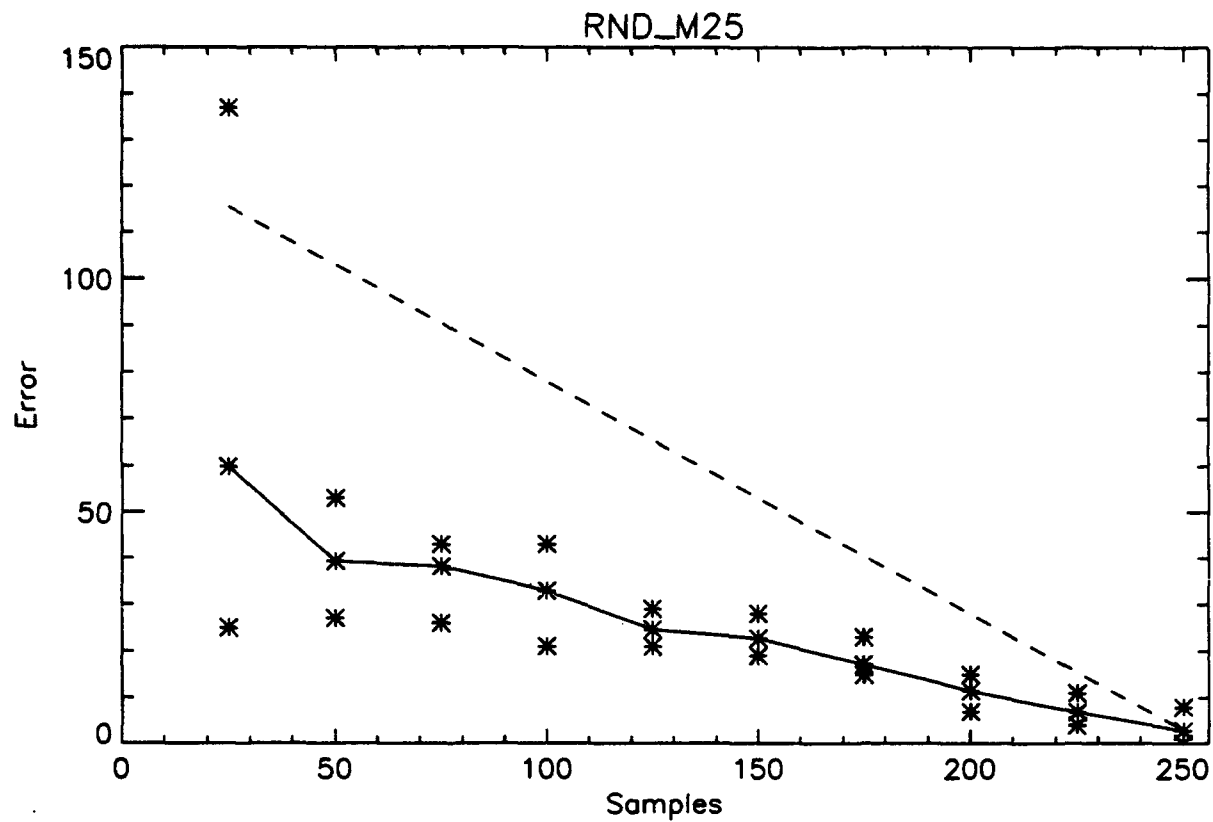
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- - - △ - - - Avg DFC

FLASH dni0e300



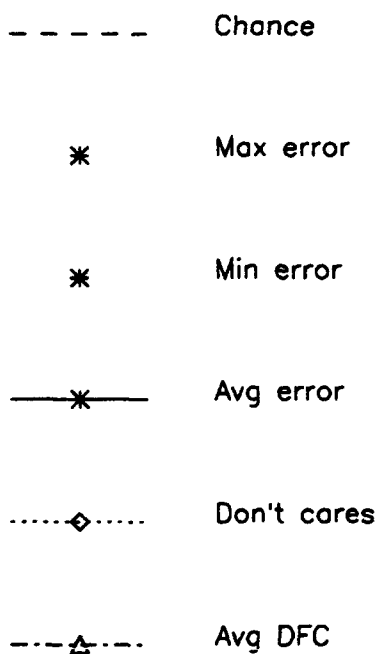
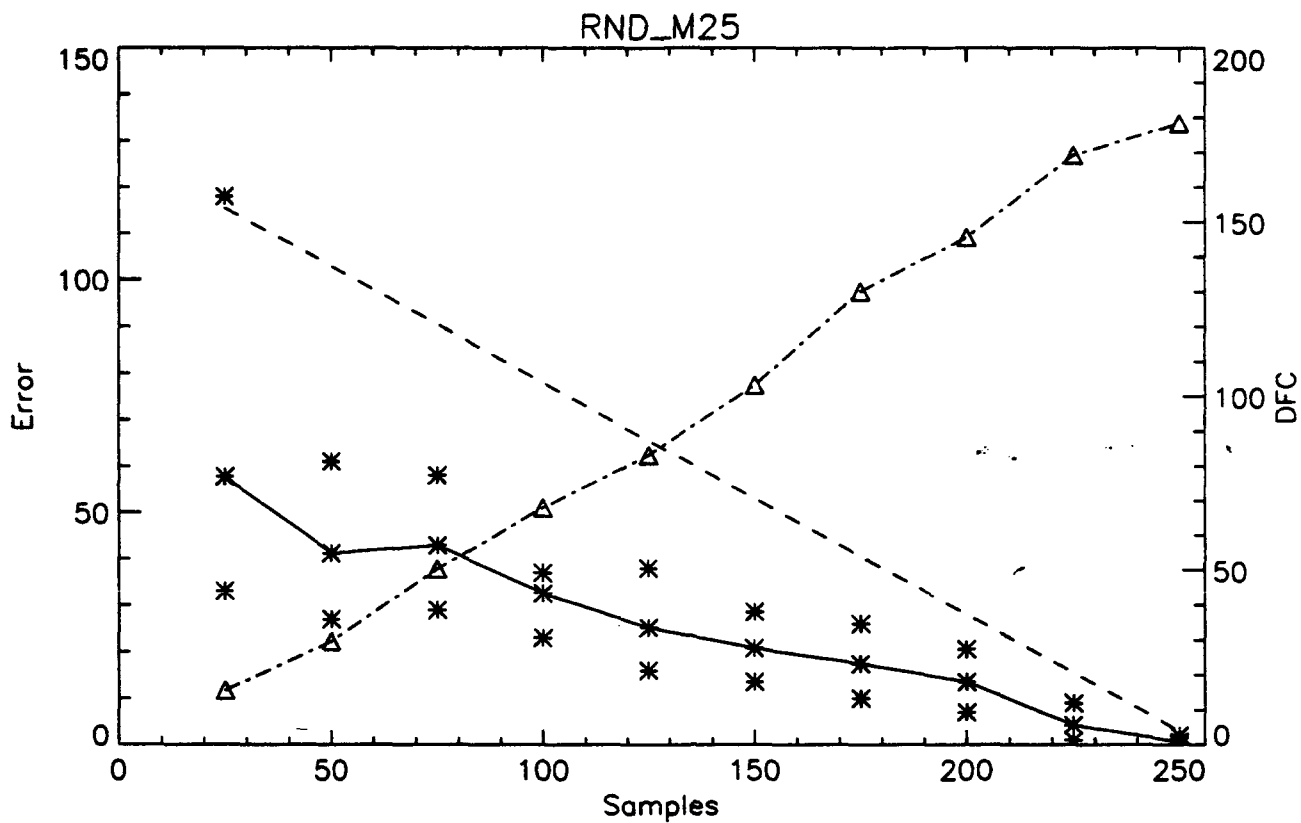
--- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

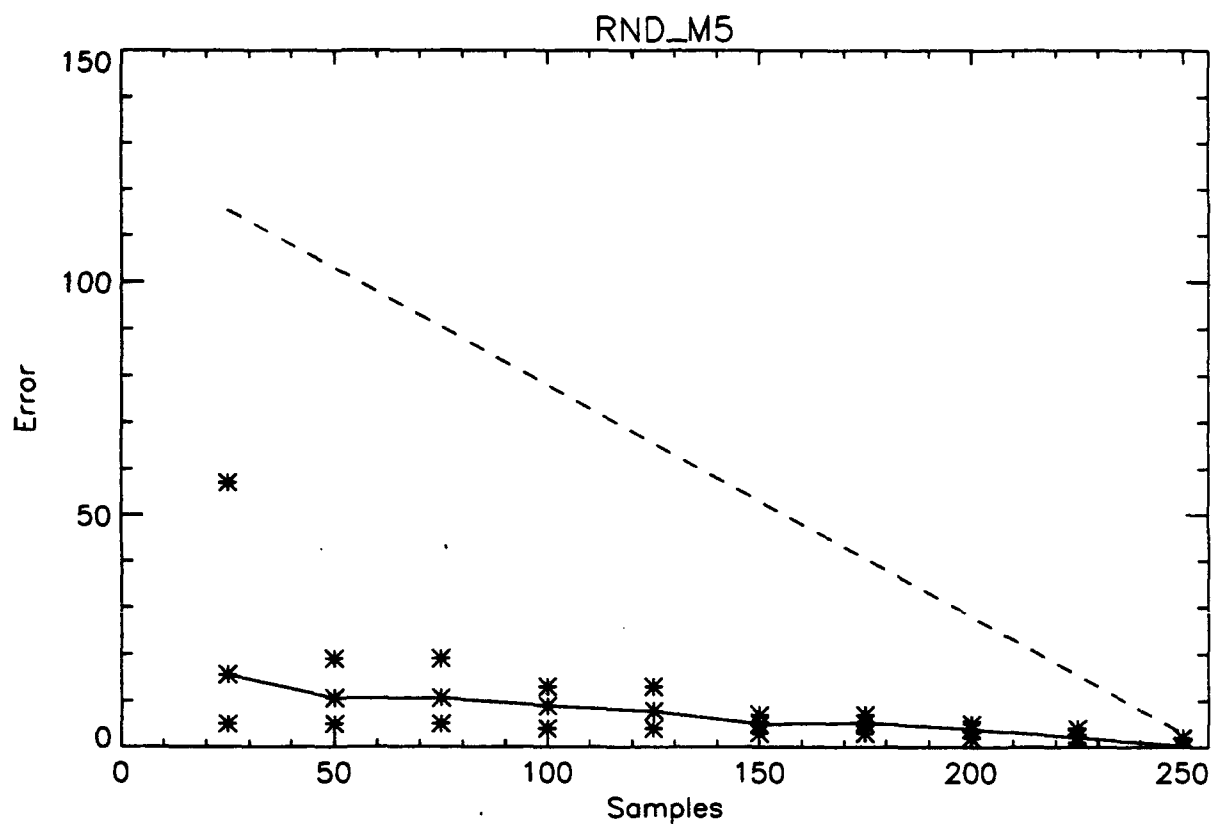
* Max Error

* Min Error

—*— Avg Error



FLASH dni0e300



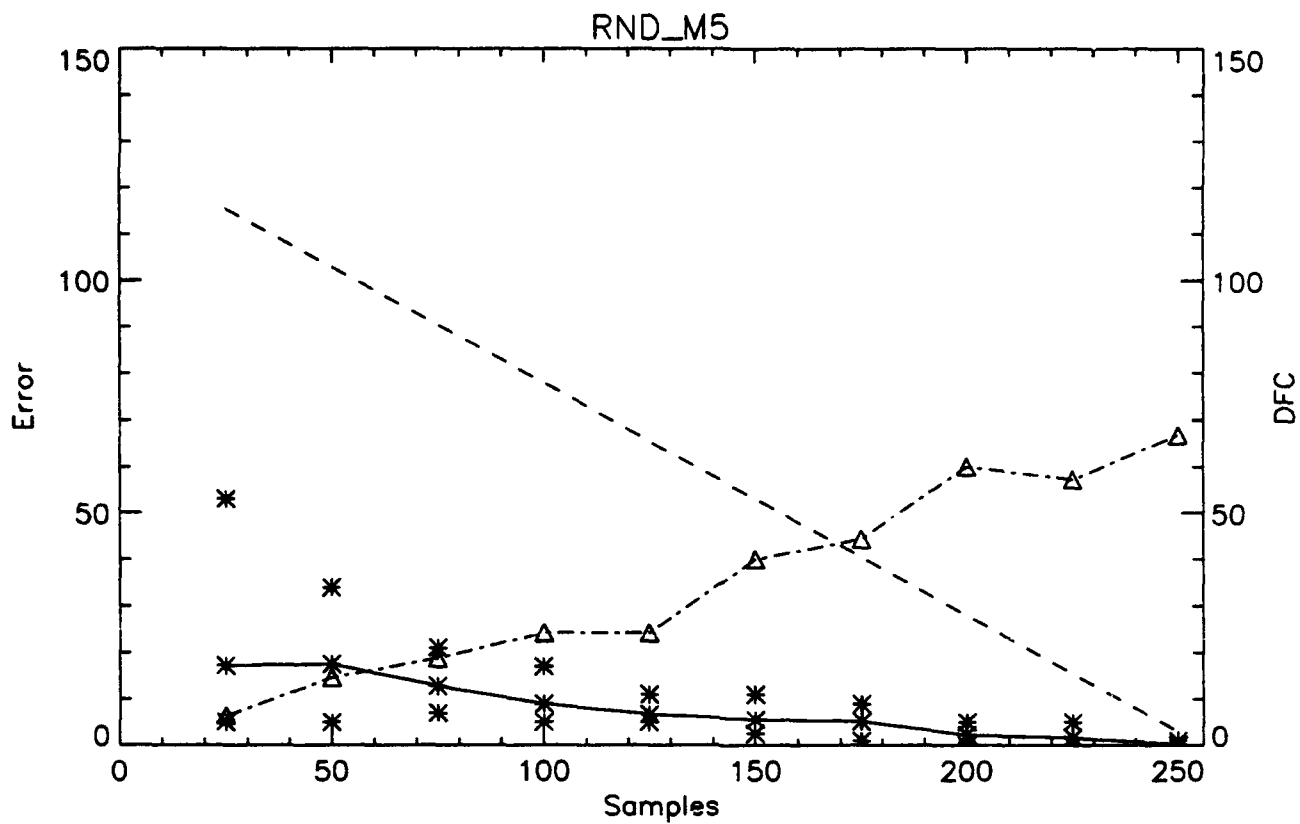
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

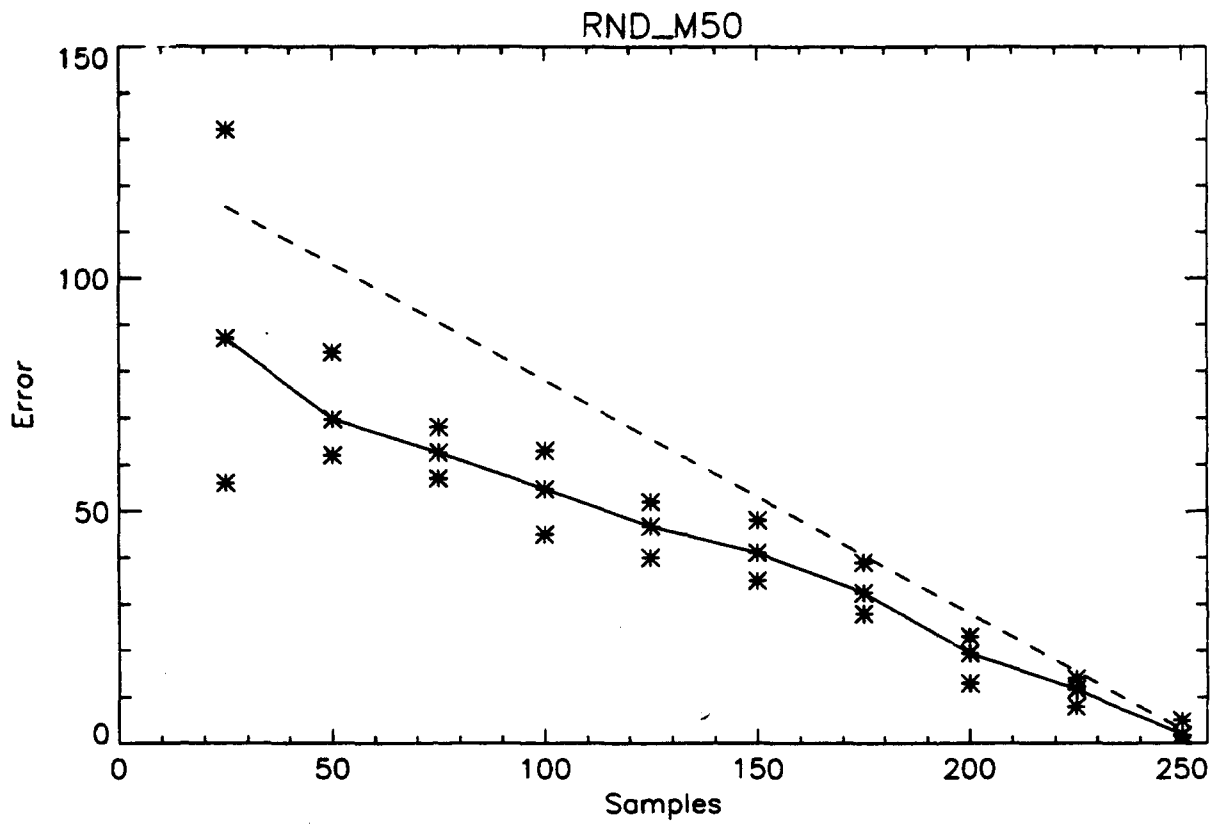
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-.-△-.-.- Avg DFC

FLASH dni0e300



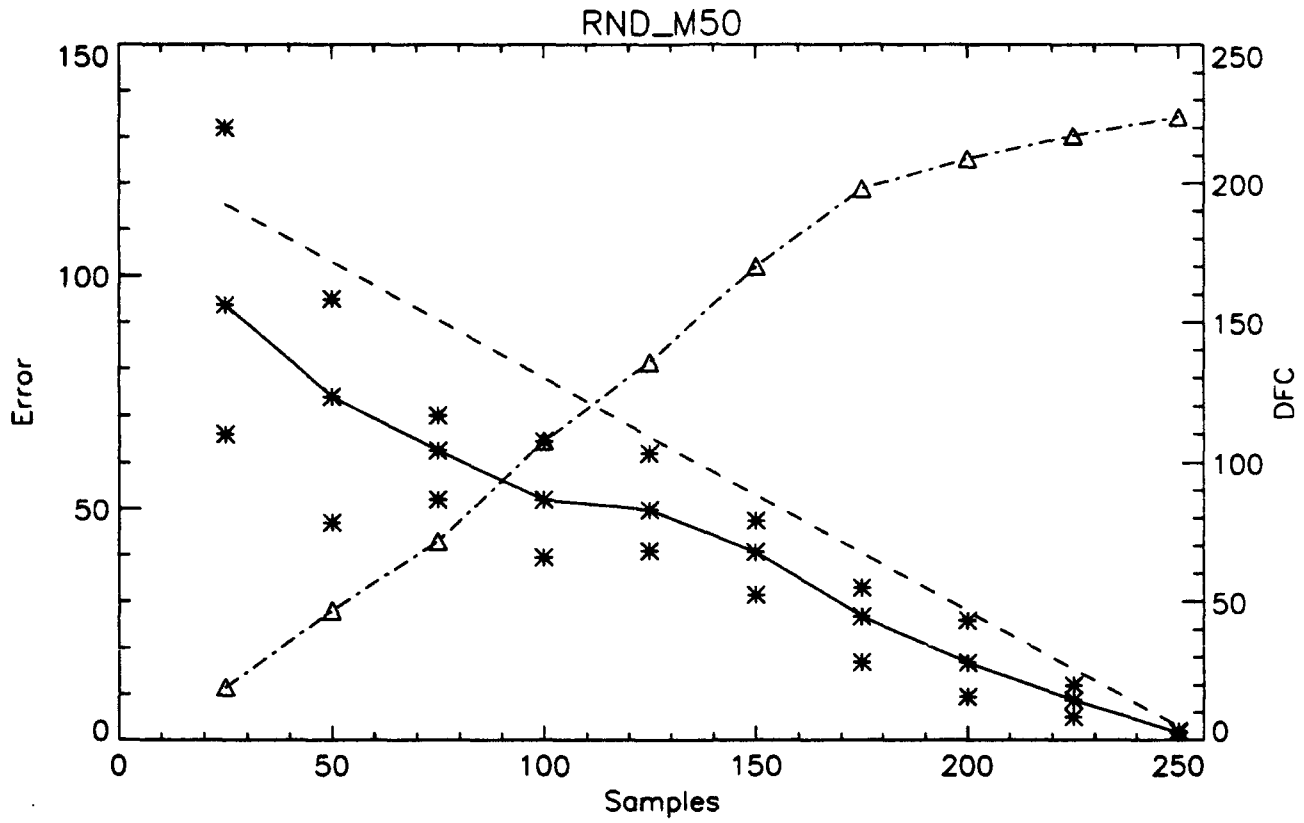
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

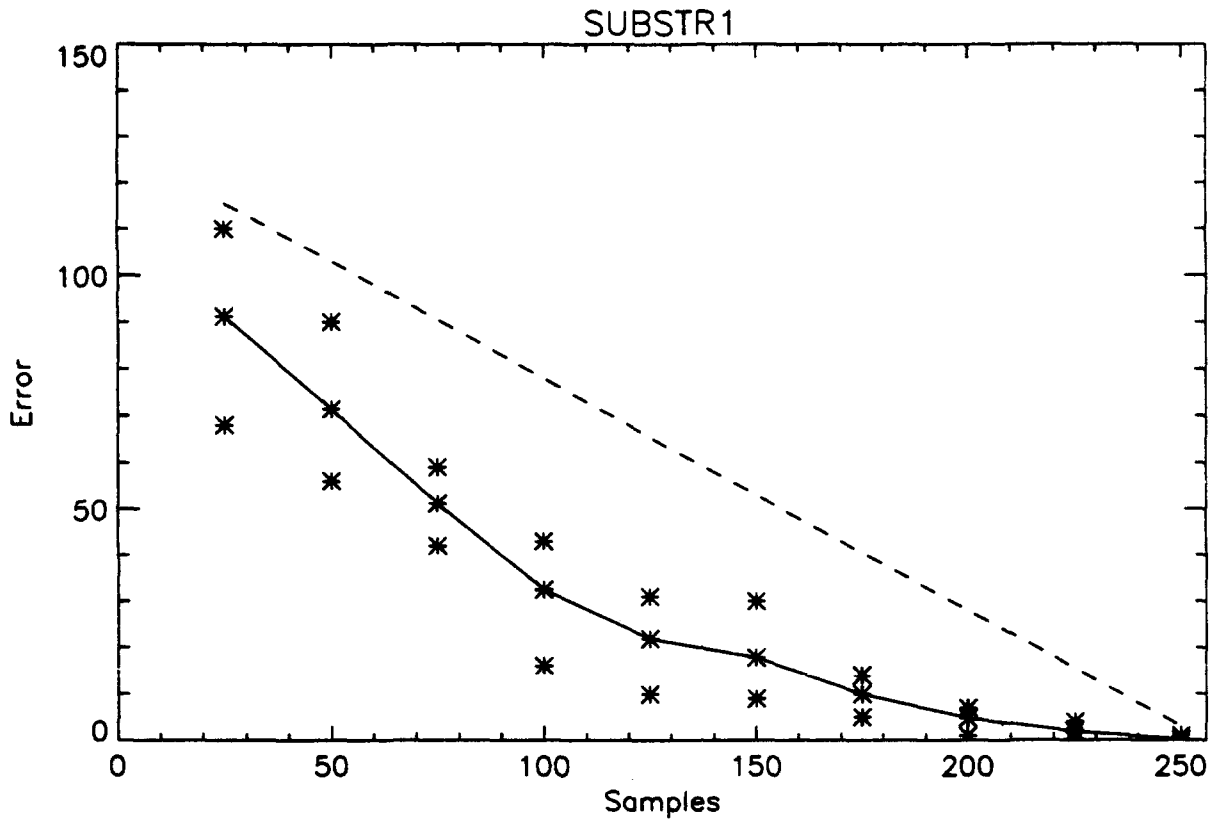
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-.-△-.-.- Avg DFC

FLASH dni0e300



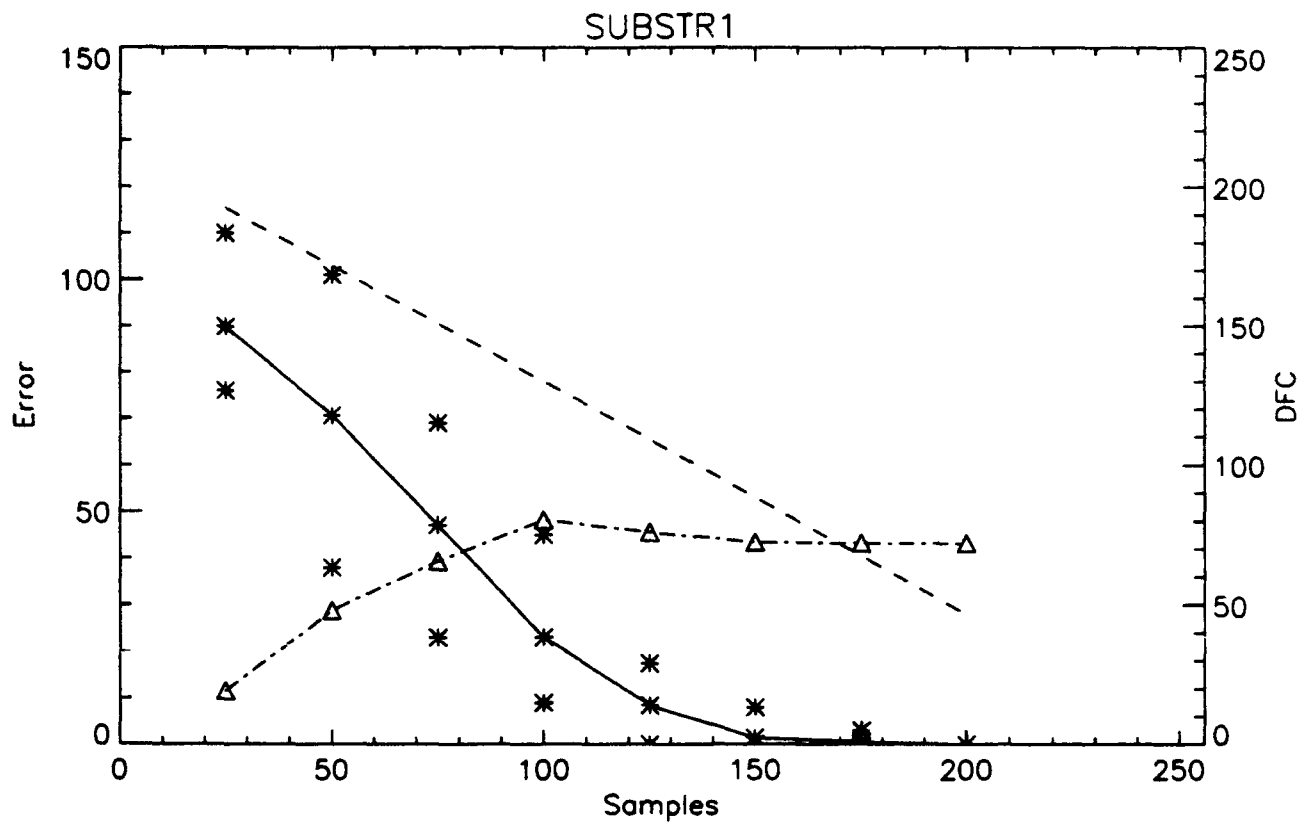
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

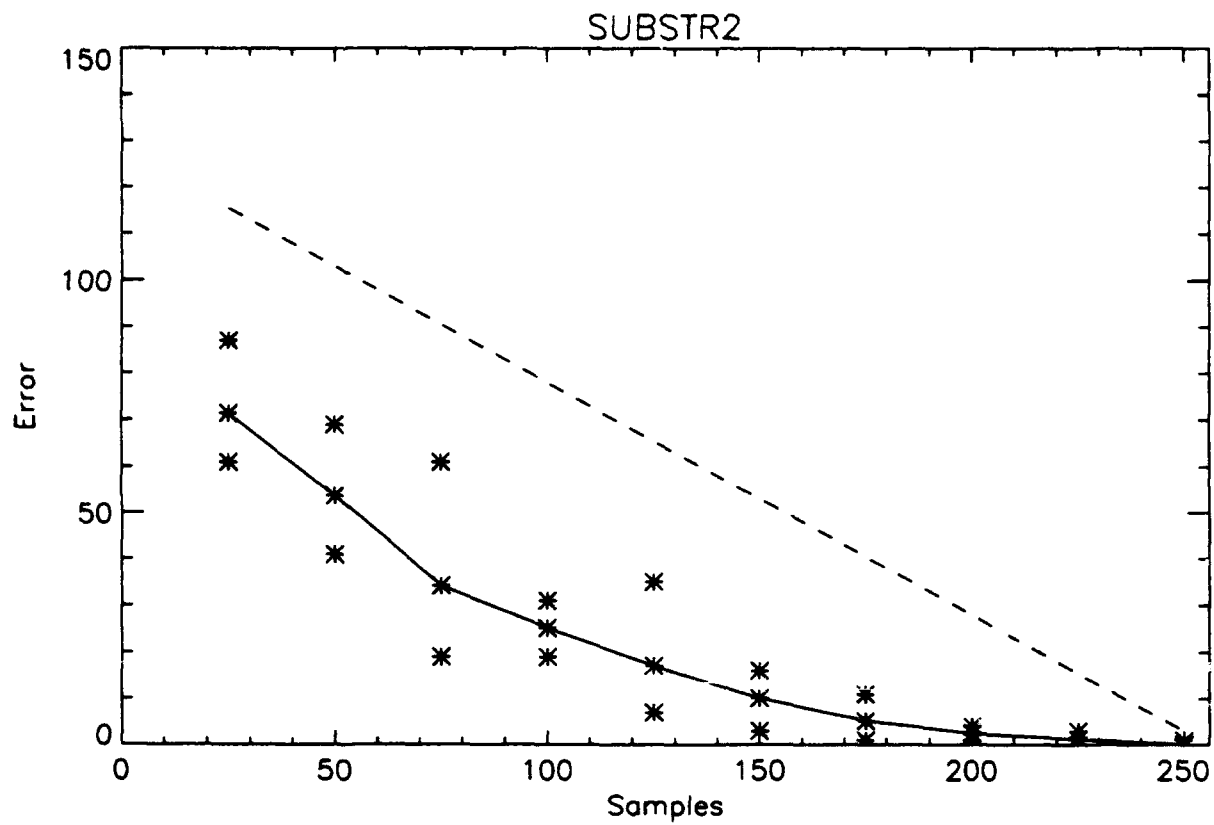
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



----- Chance

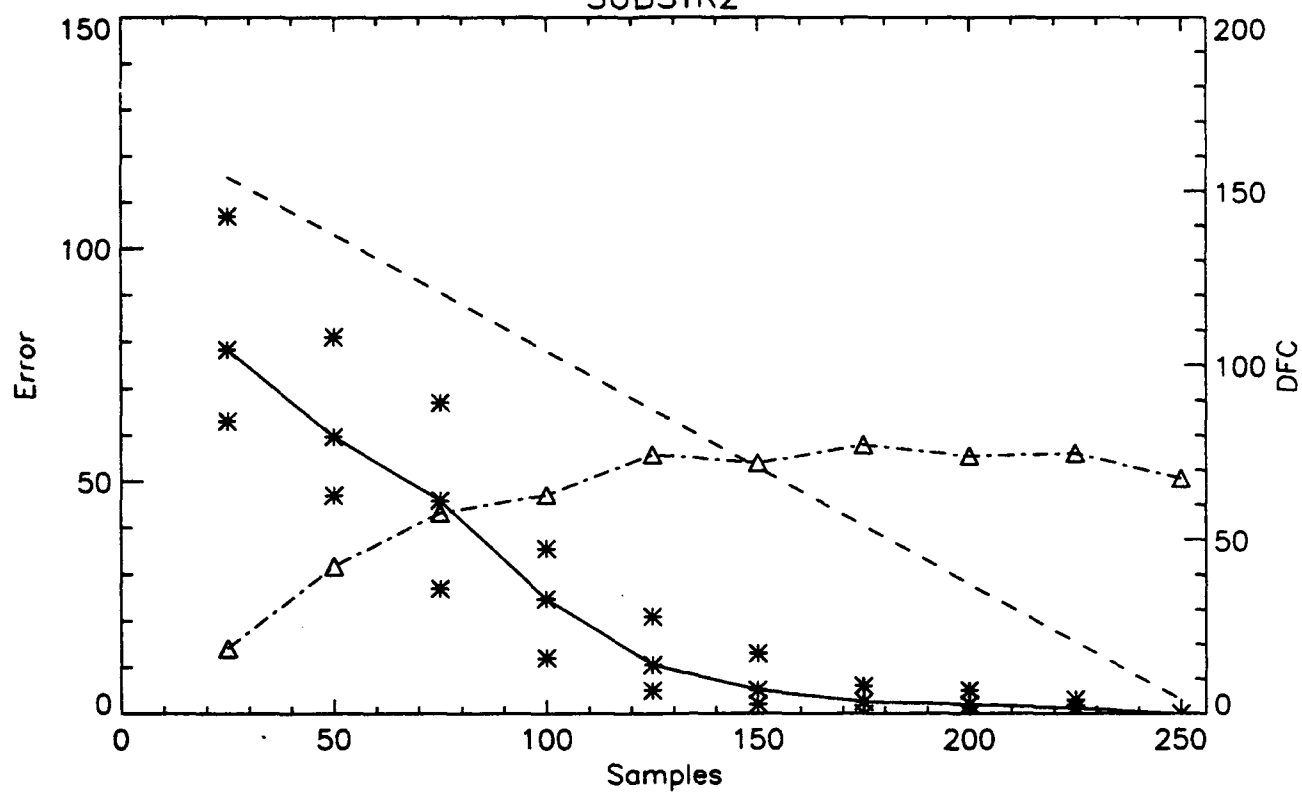
C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

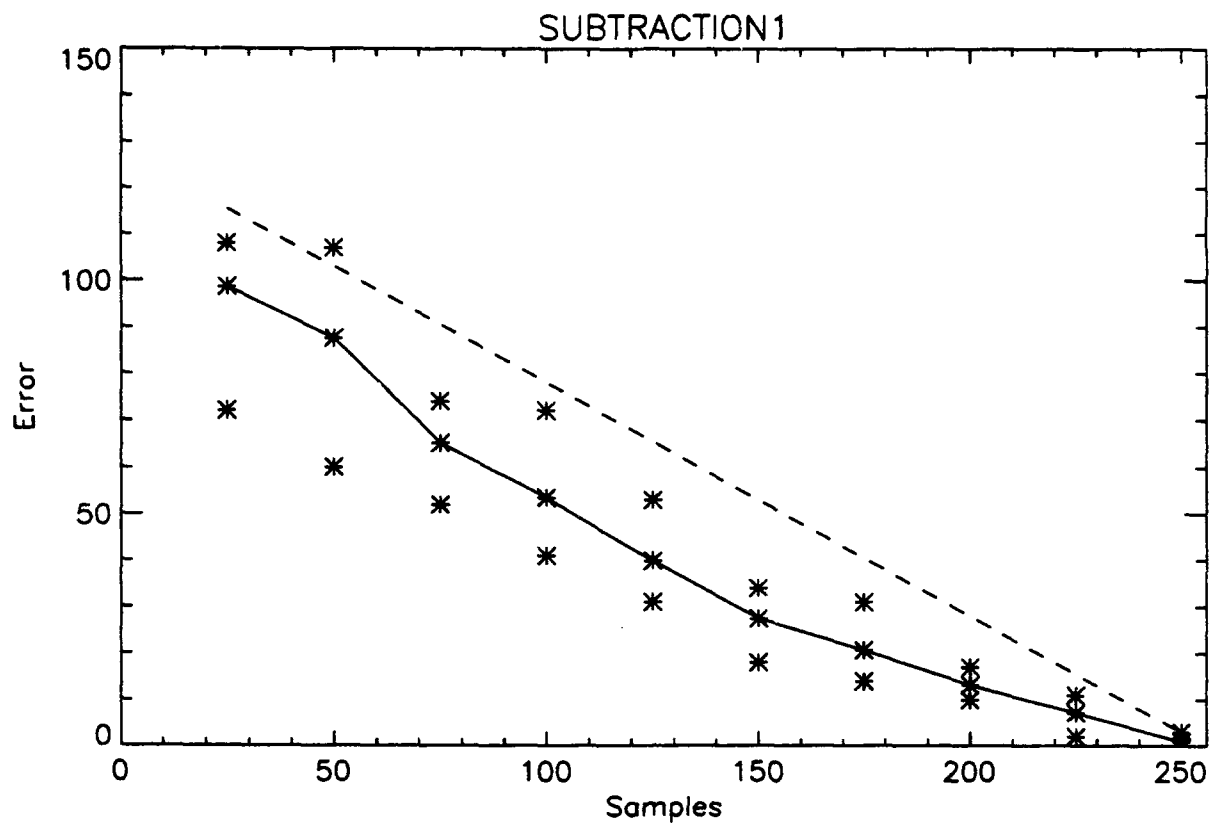
—*— Avg Error

SUBSTR2



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- .-△-.- Avg DFC

FLASH dni0e300



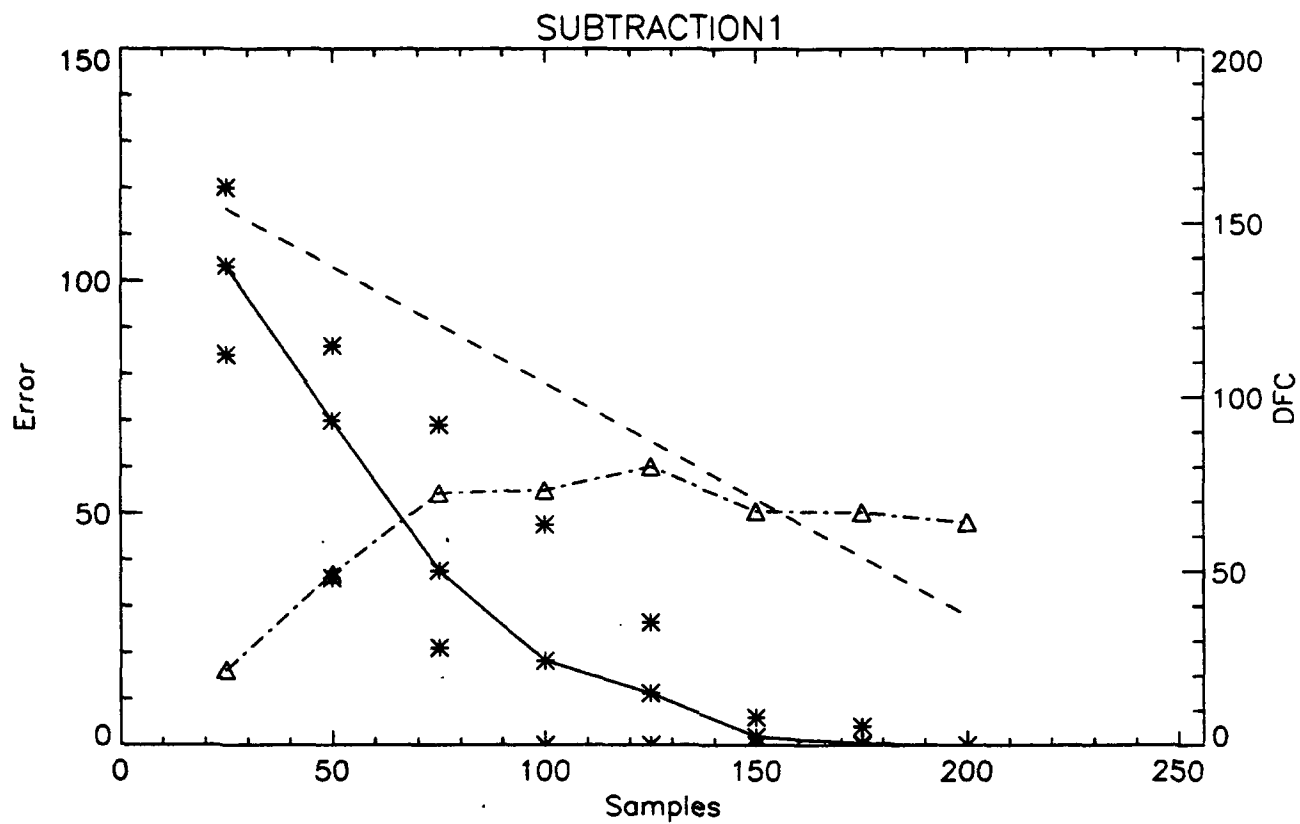
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

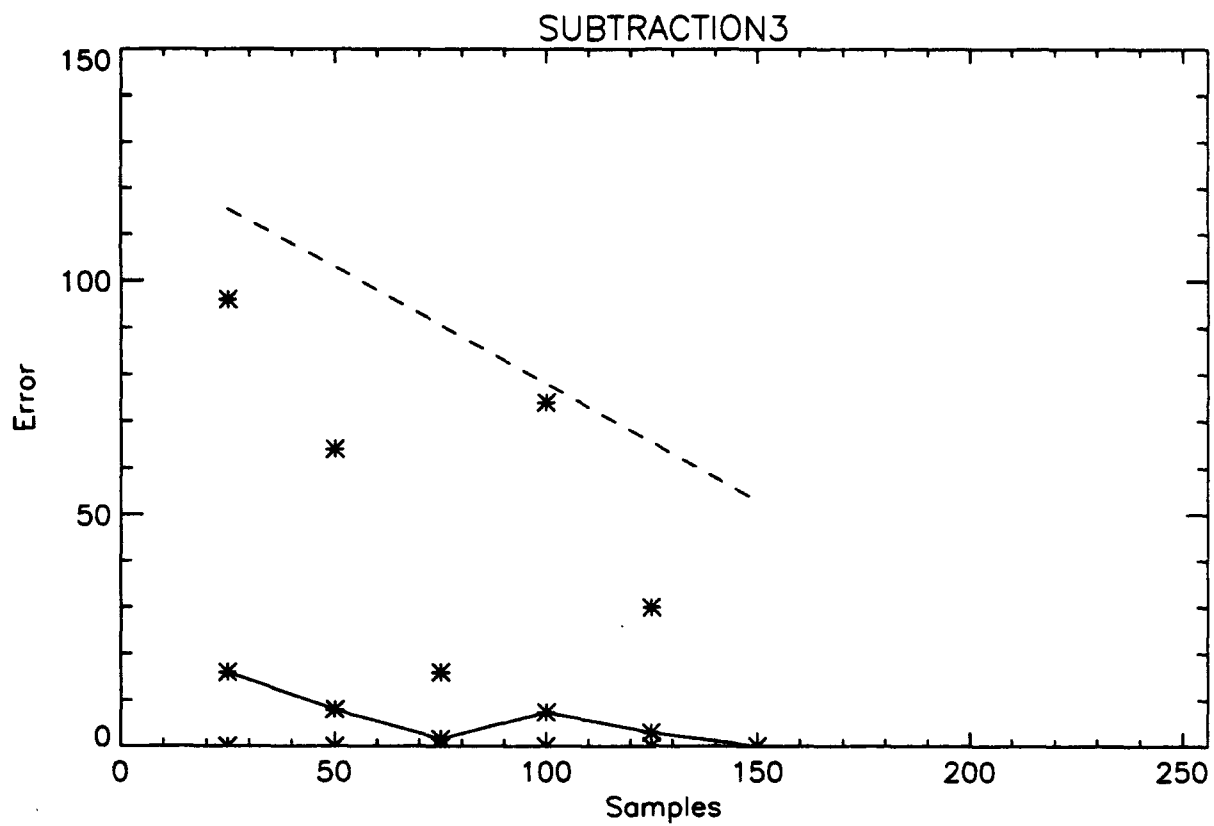
* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300



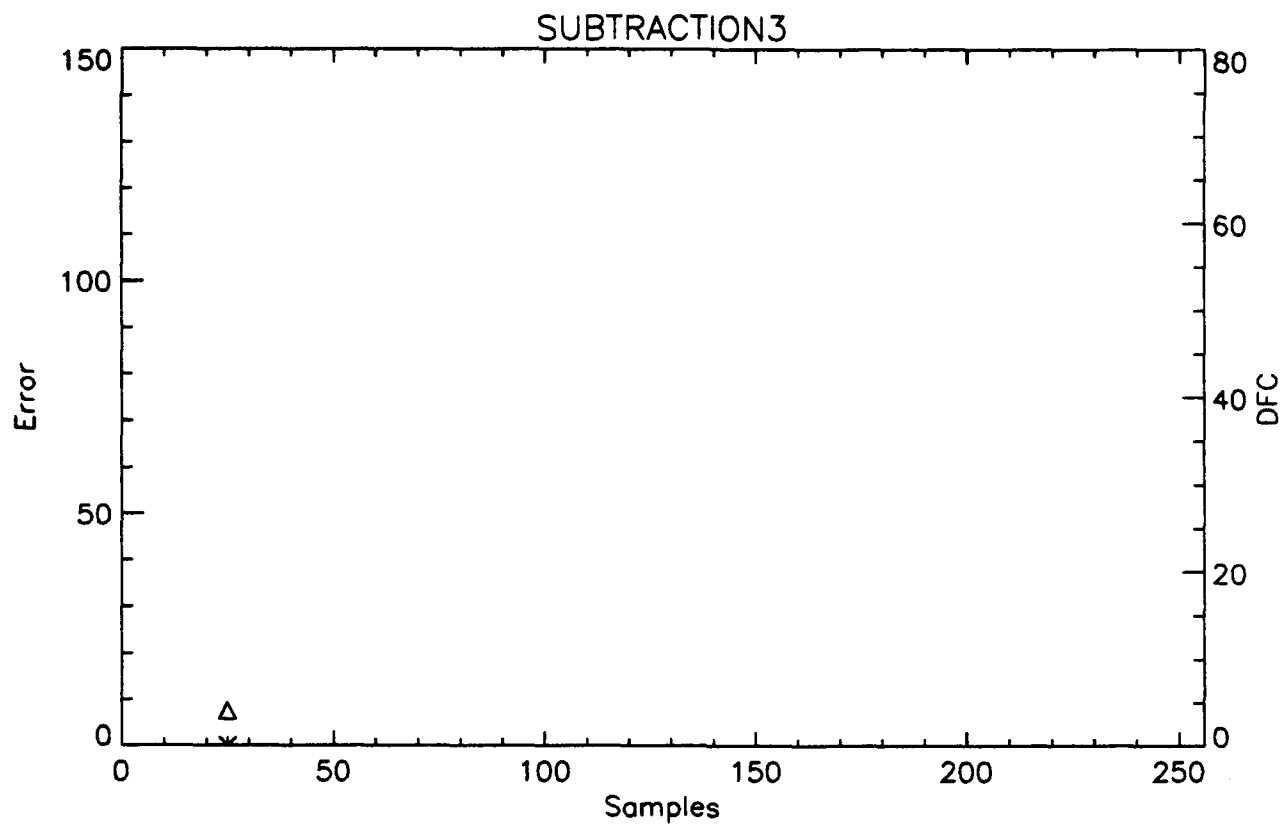
----- Chance

C4.5 Threshold=0 (-m 0), 10 Trees (-t 10)

* Max Error

* Min Error

—*— Avg Error



- Chance
- * Max error
- * Min error
- *— Avg error
-◇..... Don't cares
- △--- Avg DFC

FLASH dni0e300

B Listing of the Decomposition Plan

Appendix B shows the actual decomposition plan used in our tests with FLASH. The listing is a printout of the file dni0e300.


```

Decomp Plan:
Selection Plan:
0      = use shared variables
2      = method
0      = first part type
0      = stopping condition
Evaluation Plan:
      = no of partition tests
0      = measure challenger by
0      = measure champ by
1      = threshold in n
      = champ_multiplier
0      = dp_for_children_is_same
Decomp Plan:
Selection Plan:
0      = use shared variables
20     = method
0      = first part type
1      = stopping condition
30     = stopping condition parameter
Evaluation Plan:
2      = no of partition tests
4      = measure challenger by
1      = measure champ by
4      = threshold in n
1      = champ_multiplier
1      = measure challenger by
1      = measure champ by
4      = threshold in n
1      = champ_multiplier
1      = Random No generator seed (>0)
0      = dp_for_best_part_children_is_same
Decomp Plan:
Selection Plan:
0      = use shared variables
12     = method
2      = first part type
1      = stopping condition
1      = stopping condition parameter
Evaluation Plan:
1      = no of partition tests
4      = measure challenger by
4      = measure champ by
4      = threshold in n
1      = champ_multiplier
1      = Random No generator seed (>0)
1      = dp_for_best_part_children_is_same
1      = Random No generator seed (>0)
1      = dp_for_best_part_children_is_same

```